PHILIPS

LOGIC ANALYZERS PM 3580/PM 3585

SCPI programming manual

FLUKE AND PHILIPS - THE GLOBAL ALLIANCE IN TEST & MEASUREMENT

PHILIPS

PHILIPS

Guarantee Statement

This Philips guarantee is in addition to all rights which the buyer may have against his supplier under the sales agreement between the buyer and the supplier and according to local legislation.

Philips guarantees this product to be free from defects in material and workmanship under normal use and service for a period of one (1) year from the date of shipment. This guarantee does not cover possible required re-calibration and/or standard maintenance actions. This guarantee extends only to the original purchaser and does not apply to fuses, batteries, or to any product or part thereof that has been misused, altered or has been subjected to abnormal conditions of operation and handling.

Fluke/Philips-supplied software is guaranteed to be properly recorded on non-defective media. We will replace improperly recorded media without charge for 90 days after shipment upon receipt of the software. Our software is not guaranteed to be error free.

Philips' obligation under this guarantee is limited to have repaired or replace a product that is returned to an authorized Philips Service Centre within the guarantee period, provided that Philips determines that the product is defective and that the failure has not been caused by misuse, alteration or abnormal operation.

Guarantee service for products installed by Philips will be performed at the Buyer's facility at no charge within Philips' service travel area; outside this area guarantee service will be performed at the Buyer's facility only upon Philips' prior agreement and the Buyer shall pay Philips round trip travel expenses.

If a failure occurs, send the product, freight prepaid to the Service Centre designated by Philips with a description of the difficulty. At Philips' option, repairs will be made or the product will be replaced. Philips shall return the product, F.O.B. Repair Centre, transportation prepaid, unless the product is to be returned to another country, in which case the Buyer shall pay all shipping charges, duties and taxes. Philips assumes NO risk for damage in transit.

Disclaimer

The foregoing guarantee is exclusive and is in lieu of all other guarantees, expressed or implied, including but not limited to any implied guarantee of merchantability, fitness, or adequacy for any particular purpose or use. We shall not be liable for any direct, indirect, special incidental, or consequential damages, whether based on contract, tort, or otherwise.

Some countries or states do not allow the foregoing limitations. Other rights may also vary.

© Copyright Philips Electronics N.V. 1992
Printed in the Netherlands

Preface

Thank you for purchasing the PF 8653/x0 Remote Programming option for the PM 3580/PM 3585 Logic Analyzers.

Should you have any comments on how this product could be improved then please contact your local Fluke/Philips representative. Fluke/Philips addresses are listed in chapter 11 of the PM 3580/PM 3585 Logic Analyzers User Manual.

Table of Contents

Guarantee Statement iii
Disclaimer iv
Preface v
Introduction 1
Chapter Overview 1-2
References 1-3
The SCPI Standard 2
Introduction to SCPI 2-2
The History behind SCPI 2-2
Advantages of SCPI 2-3
The SCPI Consortium 2-3
Related Standards under SCPI 2-4
Introduction to IEEE-488.2 2-5
Message Exchange Control Protocol 2-5
Syntax Definitions 2-7
Common Program Messages 2-8
Status Reporting 2-9
SCPI Messages 2-10
Syntax and Style 2-11
Syntax of Program Messages 2-11
Command Tables 2-16
Command Parsing 2-17
Syntax of Response Messages 2-19
Parameters 2-20
The Instrument Model 3
General Instrument Model 3-2
Subsystems 3-3
Analyzer Instrument Model 3-6
Command Description 4
Introduction 4-4
Syntax Notation 4-5
Standard Syntactical Items 4-6
Other Syntactical Items 4-8
DISPlay Subsystem 4-9
:DISPlay:ENABle 4-11
:DISPlay:INPut 4-13
:DISPlay:MESSage 4-15
:DISPlay:QUEStion 4-17

FORMat Subsystem 4-19
:FORMat:DATA 4-21
:FORMat:SREGister 4-23
:FORMat:STAMp 4-25
INPut Subsystem 4-27
:INPut:GROup:THReshold 4-29
MEMory Subsystem 4-33
:MEMory:CLEar:MEASurement 4-35
:MEMory:COPY:MEASurement 4-36
:MEMory:DUMP:MEASurement 4-37
:MEMory:EXCHange:MEASurement 4-39
:MEMory:TABLe:ACCess:FORWard 4-40
:MEMory:TABLe:CHANlist 4-42
:MEMory:TABLe:COMPare:SKEW 4-43
:MEMory:TABLe:POINts 4-46
:MEMory:TABLe:POSition:NUMBer 4-47
:MEMory:TABLe:POSition:POINt 4-49
:MEMory:TABLe:POSition:STAMp 4-51
:MEMory:TABLe:POSition:VALue 4-53
:MEMory:TABLe:SAMPle:DATA 4-56
:MEMory:TABLe:SAMPle:DURation 4-59
:MEMory:TABLe:SAMPle:STAMp 4-61
:MEMory:TABLe:SAMPle:STATus 4-63
:MEMory:TABLe:SAMPle:VALue 4-65
:MEMory:TABLe:SAMPle:SELect 4-67
MMEMory Subsystem 4-69
:MMEMory:CATalogue 4-71
:MMEMory:COPY 4-73
:MMEMory:DELete 4-74
:MMEMory:INITialize 4-75
:MMEMory:LOAD:CONFiguration 4-76
:MMEMory:LOAD:MEASurement 4-77
:MMEMory:LOAD:SYMBols 4-79
:MMEMory:LOAD:SYSTem 4-80
:MMEMory:MOVE 4-81
:MMEMory:STORe:MEASurement 4-82
:MMEMory:STORe:SCReen 4-84
:MMEMory:TRANsfer 4-85
SENSe Subsystem 4-87
:SENSe:CHANnel:ACTivity 4-89
STATus Subsystem 4-91

*CLS 4-93
*ESE 4-94
*ESR 4-96
*SRE 4-97
*STB 4-98
:STATus:OPERation:CONDition 4-99
:STATus:OPERation:ENABle 4-100
:STATus:OPERation:EVENt 4-101
:STATus:OPERation:NTRansition 4-102
:STATus:OPERation:PTRansition 4-103
:STATus:PRESet 4-104
:STATus:QUEStionable:CONDition 4-105
:STATus:QUEStionable:ENABle 4-106
:STATus:QUEStionable:EVENt 4-107
:STATus:QUEStionable:NTRansition 4-108
:STATus:QUEStionable:PTRansition 4-109
:STATus:QUEue:NEXT 4-110
SYNChronization Subsystem 4-111
*OPC 4-113
*WAI 4-114
SYSTem Subsystem 4-115
*IDN 4-117
*OPT 6-118
*RST 4-119
:SYSTem:BOOT 4-120
:SYSTem:COMMunicate:GPIB:ADDRess 4-121
:SYSTem:DATE 4-123
:SYSTem:ERRor 4-125
:SYSTem:KEY 4-126
:SYSTem:KLOCk 4-128
:SYSTem:PRESet 4-129
:SYSTem:TIME 4-130
:SYSTem:VERSion 4-132
TEST Subsystem 4-133
*TST 4-135
TRIGger Subsystem 4-137
:ABORt 4-139
:INITiate 4-140
:STOP 4-141
Status Reporting System 5
Analyzer Status 5-2

Standard Status 5-6
ESR register 5-7
ESE register 5-8
STB register 5-8
SRE register 5-10
Operation/Questionable Status 5-11
Condition Registers 5-12
Condition Filters 5-13
Event Register 5-14
Event Enable Register 5-14
Error and Event Reporting 6
Structure 6-2
Errors 6-4
Command Errors 6-4
Execution Errors 6-9
Device dependent Errors 6-12
Query Errors 6-13
Panel Event Handling 7
Panel Code Queue 7-2
Panel Codes 7-3
Button Codes 7-4
Dialknob Codes 7-5
Panel Simulation 7-6
Command Synchronization 8
Wait to Complete 8-3
Operation Complete 8-4
Acquisition Control 9
Acquisition State 9-2
Operation Condition 9-4
Synchronization 9-5
Acquisition Data Access 10
Measurement Structures 10-2
Memory Tables 10-5
Table Selection 10-6
Table Points 10-7
Table Position 10-7
Table Entry 10-10
Remote/Local Operation 11
Remote Operation 11-2
Local Operation 11-3
Operator Communication 11-4

RS-232 Interface 12
Analyzer Setup 12-2
Interface 12-3
Signals 12-3
Cabling 12-4
Data Format 12-5
Handshaking 12-6
Data Encoding 12-6
Limitations 12-7

Chapter 1
Introduction

Chapter Overview 1-2
References 1-3

The PF8653/20 and PF8653/30 remote programming options allow you to program and control a PM 3580/PM 3585 Logic Analyzer via respectively an IEEE-488.2 and RS-232 interface. These options require the PF8690/xx Version 2.0 system software being installed.

Chapter Overview

This remote control facility is based upon SCPI (Standard Commands for Programmable Instruments) which owes much of its structure and command syntax to the IEEE-488.2 standard. A short overview of these standards and the relation between them is given in Chapter 2: The SCPI Standard.

In order to facilitate remote programming and to achieve compatibility between the various remote programmable Test & Measurement instruments, SCPI defines a generalized instrument model. This model, as well as the model of the Logic Analyzer is described in Chapter 3: The Instrument Model.

Chapter 4, Command Descriptions, describes the syntax and semantics of all commands and queries supported and implemented in the software of the analyzer. This includes the IEEE-488.2 and SCPI mandatory commands.

SCPI defines an extensive status reporting model in order to show at any time the current state of the instrument and to report all kinds of events that caused a change in its state. The functions provided by this model for the Logic Analyzer are described in Chapter 5: The Status Reporting System.

Chapter 6, Error and Event Reporting, describes the way errors are reported to the control program. It contains a list of all possible errors that may occur as a result of the receipt of a command or query.

Chapter 7, Panel Events, describes the way the control program may be kept informed about the analyzer's front panel events; i.e. the the keying and dial operations at the front panel. In addition a description is given how the controller can simulate front panel input.

The facilities provided to synchronize the controller program with the execution of commands and queries on the analyzer is described in Chapter 8: Command Synchronization.

Chapter 9, Acquisition Control, describes how the control program can control the acquisition with commands provided by the Trigger Subsystem and how the controller can be kept informed about the current acquisition state and state transitions via the Status Reporting System.

The way acquisition data is organized and recorded in memory, as well as the facilities provided to access the data is described in Chapter 10: Acquisition Data Access.

Chapter 11, Remote/Local Operation, describes certain aspects of simultaneous access to the analyzer and how the analyzer can be setup for either remote or local operation. In addition a description is given how the controller can communicate with the local operator.

Those aspects of remote control, which only apply to RS-232 are described in Chapter 12, RS-232 Interface. This includes setup of the analyzer for operation via RS-232, the RS-232 cabling and transmission characteristics and some limitations in the usage of this remote programming option compared with the IEEE interface.

References

The following documents provide additional information regarding remote operation via IEEE and RS-232 interfaces:

- PM 3580/PM 3585 Getting Started Guide
- PM 3580/PM 3585 User Manual
- PM 3580/PM 3585 Reference Guide
- ANSI/IEEE Standard 488.1-1987, Digital Interface for Programmable Instrumentation.

Published by The Institute of Electrical and Electronics Engineers

ISBN 471-62222-2

- ANSI/IEEE Standard 488.2-1987, Codes, Formats, Protocols, and Common Commands.

  Published by The Institute of Electrical and Electronics Engineers

  ISBN 471-61871-3

- EIA Standard, Interface Between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Interchange (RS-232).

  Published by Electronic Industries Association

- SCPI 1992 Volume 1: Syntax and Style

  Published by SCPI Consortium

- SCPI 1992 Volume 2: Command Reference

  Published by SCPI Consortium

- General Purpose Interface Bus (GPIB) IEC 625-1 IEEE-488.1

  Published by Philips Electronics N.V., T&M.

  Publication number: 4822 872 80148

- SCPI - Standard Commands For Programmable Instruments

  Published by Philips Electronics N.V., T&M.

  Publication number: 4822 872 80149

- PM 2201 Installation of the GPIB Interface

  Published by Philips Electronics N.V., T&M

  Publication number: 4822 872 80124

- PM 2201 - PM 2202 GPIB IO Software Drivers

  Published by Philips Electronics N.V., T&M

  Publication number: 4822 872 80126

- PM 2201 - PM 2202 IEEE-488.2 Software Drivers

Published by Philips Electronics N.V., T&M

Publication number: 4822 872 80127

- PM 2201 - PM 2202 Getting Started with Software GPIB Drivers

  Published by Philips Electronics N.V., T&M

  Publication number: 4822 872 80123

Chapter 2
The SCPI Standard

Introduction to SCPI 2-2
- The History behind SCPI 2-2
- Advantages of SCPI 2-3
- The SCPI Consortium 2-3
- Related Standards under SCPI 2-4

Introduction to IEEE-488.2 2-5
- Message Exchange Control Protocol 2-5
- Syntax Definitions 2-7
- Common Program Messages 2-8
- Status Reporting 2-9

SCPI Messages 2-10
- Syntax and Style 2-11
- Syntax of Program Messages 2-11
- Command Tables 2-16
- Command Parsing 2-17
- Syntax of Response Messages 2-19
- Parameters 2-20

Introduction to SCPI

SCPI (Standard Commands for Programmable Instruments) is a standardized set of commands to be used for remote control of programmable test and measurement instruments. It defines the syntax and semantics that the controller must use to communicate with an instrument, and is implemented in the instrument soft or firmware.

This chapter gives an overview of SCPI, as implemented in modern test and measurement instruments from most manufacturers, amongst others Philips/ Fluke instruments.

SCPI is based upon the IEEE-488.2 standard, to which it owes much of its structure and syntax. Nevertheless it has been designed as an instrument control command set that can be used with any of the standard interfaces, for example IEC 625-1/IEEE-488.1, VXI and RS-232.

The History behind SCPI

Since the introduction of the IEC 625/IEEE-488, "Digital Interface Standard for Programmable Instrumentation", many test and measurement instruments have been supplied with this interface, often referred to as the GPIB interface. The GPIB specification sets a standard for connecting instruments and controllers and for the data exchange protocol. However, it does not specify the meaning of the information sent. Instrument manufacturers were free to invent commands as they developed new instruments.

Notes:
- ANSI IEEE is the standard in the United States of America.
- IEC is the international standard, often referred to in Europe. Both standards are the same.

With the introduction of the IEEE-488.2-1987 "Codes, Formats, Protocols and Common Commands for use with the ANSI/IEEE Std 488.1-1987" standard, a small set of 'common program messages' were defined. These commands were intended only for general housekeeping purposes. IEEE-488.2 also describes the syntax for commands to instruments and responses from instruments. However, it does not specify the command names and command functionality.

Advantages of SCPI

SCPI defines a standard set of instrument programming command and response messages for each instrument function.

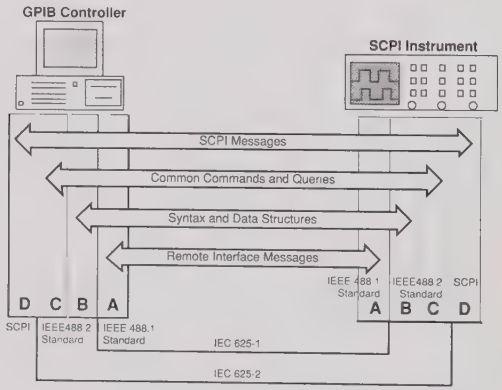
SCPI simplifies maintenance and management of the programs. Nowadays changes and additions in a good working program are difficult to make. This is because of the great diversity of program messages and instruments. Programs are often difficult to understand for anyone other than the original programmer. Because of his knowledge of the standard, a programmer with SCPI experience will quickly understand the meaning and functions of an SCPI program. Changes, extensions and additions in an existing application program are much easier to achieve. SCPI is a step forwards towards portability of instrument programming software. As a consequence it facilitates the exchange of instruments with little or no changes in the application program, provided of course that the exchanged instruments have the same functionality.

The SCPI Consortium

The industry standard "SCPI" is published by an international consortium, which consists of all major manufacturers of test and measurement instruments in the world. Philips and Fluke are both a sponsor and founding member of the SCPI consortium.

Related Standards under SCPI

SCPI builds on the IEEE-488.1 (IEC 625-1) and IEEE-488.2 (IEC 625-2) standards as the following diagram shows:



Standards under SCPI

This figure shows the hierarchical levels of communication between an SCPI instrument and the GPIB controller. That is from the lowest level (level A), such as handshaking on the GPIB, addressing and de-addressing of instruments, up to the highest level (level D), which are the SCPI defined commands and responses.

Introduction to IEEE-488.2

The IEEE-488.2 standard does not replace the IEEE-488.1 standard. It is additional and complementary to the IEEE-488.1 standard. Instruments that are compatible with IEEE-488.2 must also be compatible with IEEE-488.1, however IEEE- 488.1 compatible instruments may or may not conform to the IEEE-488.2 standard. The IEC 625-2 and IEEE-488.2 standards deal with several aspects of standardization of higher interface layers within instrumentation systems. The scope of these international standards is restricted to data communication between a device and the controller over a GPIB interface.

Message Exchange Control Protocol

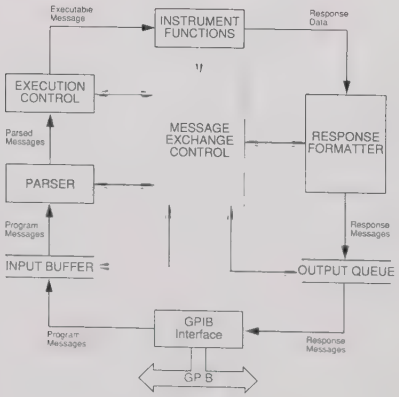
Another important part of the IEEE-488.2 standard is the Message Exchange Control protocol. This protocol precisely specifies the interactions between the several functional elements that exist between the GPIB functions and the device specific functions. In fact it specifies how messages are to be exchanged between controller and device. For example, it specifies how a device shall handle program and response messages that are received from and sent to a controller. IEEE-488.2 introduces the idea of commands and queries; queries are program messages that require the device to send a response. When the controller does not read the response to a previously sent query before sending a next command, the device will generate a 'Query' error. On the other hand, commands will not cause the device to generate a response. When the controller still tries to read a response, the device will also generate a 'Query' error.

The Message Exchange Control protocol also deals with the order of execution of program messages. It defines how to respond to several categories of errors, such as 'Command' Errors, 'Query' errors, 'Execution' Errors and 'Device Specific' errors. Basically, the protocol puts the demand upon the device to report any violation of the IEEE-488.2 rules to the controller, even when it is the controller that infringes these

rules. Concerning the message exchange between controller and device, violation of the following rules will result in the generation of a query error:

- The controller attempts to read a response message, before having properly terminated the previous program message.
- The controller attempts to send a new program message, before having read the entire response to a previously sent query.
- The controller attempts to read a response message, before having previously sent a query message.
- A deadlock condition occurs, i.e. infinitely waiting for each other.
- The controller sends a query that generates an indefinite response, which is not the last query program message unit in a program message.

IEEE-488.2 and SCPI compatibility applies to the functions implemented inside an instrument that control the instrument operation. The following figure gives a simplified logical model of the message flow inside the software of an instrument.



Message Exchange

When the controller sends a message to an SCPI instrument, roughly the following happens:

- The GPIB controller addresses the instrument as listener and sends a message. The receiving GPIB interface function places the message in the Input Buffer.
- The Parser fetches the message from the Input Buffer, parses (decodes) the message, and checks for the correct syntax. Incorrect syntax will cause command errors to be reported via the status system to the controller and in the Error/Event queue. Moreover, the parser will detect whether a response is required. This is the case when the input message is a query.
- The Parser will transfer the executable messages to the Execution Control block in token form (internal codes). The Execution Control block will gather the information required for the device action and will initiate the requested task at the appropriate time. Execution errors are reported via the status system and the Error/Event queue.
- The Instrument Functions block will perform the requested actions.
- If a query was received, the Instrument Functions block will transfer the response data (internal code) to the Response Formatter.
- The Response Formatter converts the internal codes to response messages according to the IEEE-488.2 syntax format, and places them in the Output Queue.
- When the GPIB interface is addressed as talker, it takes the formatted response message from the Output Queue and sends it over the GPIB to the controller.

Syntax Definitions

Another area that is covered by IEEE-488.2 is the specification of the syntax of program and response messages. Although the basic concept is not changed in respect to previous standardized syntaxes for instrumentation systems, the terminology and coding rules are somewhat changed. Based on gained experiences ideas have been adapted. For instance, compound headers have been introduced,

facilitating the coding of complex message structures. Also the definition of new types of messages, such as common program messages, considerably accommodate the demand of modern instrumentation systems. A broad range of data types can be used for parameter notation, supporting a variety of applications.

The user friendliness of the syntax is shown by the flexibility of program data. Within this data, which is sent by the controller (user), the syntax rules are less restrictive than for response data (which is sent by the device to the controller). For instance, program messages may be sent in any mixture of upper and lower case alpha characters. Decimal data may be sent to the device in any notation form (with or without decimal point, exponent, etc.), the data may be preceded by leading spaces and followed by a suffix for additional unit specification. A similar approach applies to the termination of messages; several terminators are allowed that unambiguously determine the end of a message.

Common Program Messages

Until now, no standards existed for frequently used functionalities that are commonly provided with the majority of instruments. IEEE-488.2 has made a step forward in this field by defining a range of common program messages that support these commonly used functionalities. This includes a number of program messages that have been defined to support IEEE-488.2 defined functions, e.g. the status reporting model. A minimum set of the common program messages is identified for mandatory implementation in all instruments that claim IEEE-488.2 compatibility.

With the common program messages it is possible to identify the device (manufacturer, model and type number, etc.) and to determine the available options. It is also possible to let the device perform a calibration cycle or to perform an internal self test. Other features are the common query for the current instrument settings or the determination of the completion of an operation. Program messages have been standardized to reset the device functions and to trigger device actions.

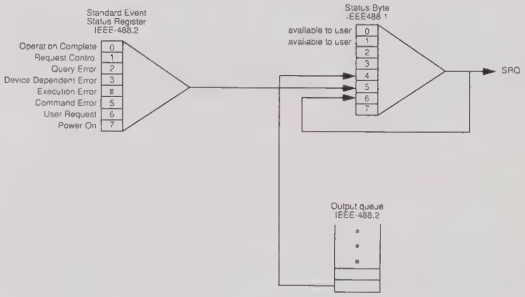
Other functions that are supported by common program messages are macros. A macro may contain complex message structures which are executed upon receipt of a simple command (macro label). Also the registers in which the current instrument settings can be saved for recall later on, can be controlled by common program messages.

Status Reporting

A lot of attention is paid to improve the status reporting structure. The GPIB standard only requires the device to send a Status Byte as the response to a serial poll. This limited requirement caused a lot of different implementations. For instance, sometimes conditions were reflected whereas other instruments reported events in the Status Byte. This caused the resetting of the Status Byte bits to occur at different moments.

The IEEE-488.2 standard issues additional requirements on the Status Byte; an overlaying status reporting structure is defined, which allows only events to be summarized in the Status Byte. Within this structure, a number of commonly reported events are standardized (e.g. several error events, power-on event, etc.). The structure facilitates the reporting of device specific events as well. This status reporting structure is supported by a number of common program messages, such as:

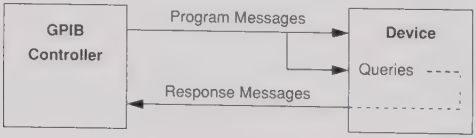
- the clearing of the status reporting structure
- for programming Enable Registers which select the events that are to be summarized in the Status Byte
- for programming the Service Request Enable Register, which selects the Status Byte bits for generation of a Service Request
- for reading the Standard Event Registers, etc..



IEEE-488.2 Status Reporting Model

SCPI Messages

The communication between the system controller and the SCPI instruments, connected to the GPIB, takes place through Program and Response Messages. A message sent from the controller to an instrument is called a Program Message. Conversely, a Response Message is the data from the instrument to the controller.



Program and Response Messages

The GPIB controller uses Program Messages to instruct the device. These can be queries or commands. The device will only send a response when explicitly requested to do so, namely when the controller sends a Query. Queries are recognized by the question mark at the end of the header, for example: *IDN? (requests the instruments identity data).

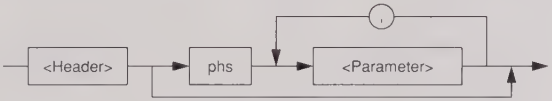
The question mark distinguishes queries from commands. A command does not cause the device to generate a response. Notice that, although there is a formal distinction between commands and program messages, a program message is often referred to as a command.

Syntax and Style

Although the fundamental message concept for program and response messages is the same, there are a number of syntactical and styling differences. These differences mainly deal with the more permissive coding of a program message and the more precise format of a response. Both the IEEE-488.2 and SCPI standard specify requirements regarding this issue.

Syntax of Program Messages

A command or query is called a program message unit. Such a program message unit consists of a header, or a header separated by a space from one or more parameters. Refer to the following syntax flow diagram:

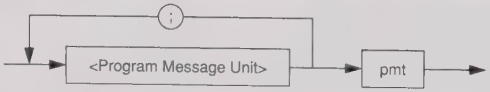


Syntax of a Program Message Unit

The phs (program header separator) between the header and the first parameter must be at least one 'white space' character. A white space is a character in the decimal range 00 up to and including 32 (space), except LF (code 10). The header consists of one or more mnemonics (key words) describing the command. The mnemonics in a command are separated by a colon (:). The parameters in a message unit are also referred to as 'Data Elements'. They are mutually separated by a comma (,), which is referred to as 'Data Separator'. Furthermore the following rules are valid:

- Any one of the 'white space' characters (dec. 0 .. 9, 11 .. 32) may be placed in between the syntactical elements of a message, e.g.:
  - precede a header
  - precede the message terminator
  - be placed in between the header and a parameter
  - be placed around a message separator
- A parameter which is specified as Decimal Numeric Data, may use different numeric representation formats. For example with or without decimal point and exponential notation.
- If a parameter has a greater precision than the device can internally handle, the device must round the value (not truncate it) before it is interpreted.

One or more program message units may be sent within a single program message. Program message units are separated by a semicolon (;). A message of one or more units is terminated by a program message terminator (pmt) as shown in the next diagram:



Syntax of a Terminated Program Message

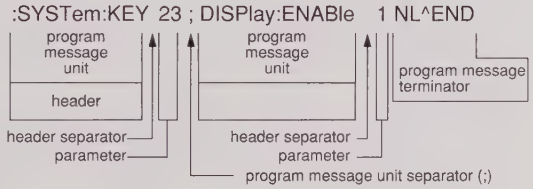
The pmt (program message terminator) must be one of the following codes:

- NL^END — This is the <new line> code sent concurrently with the END message on the GPIB.
- NL — This is the <new line> code.
- <dab>^END — This is the END message sent concurrently with the last data byte <dab>.

Notes:
- NL is the same as the ASCII LF (<line feed> = 10 decimal).
- The END message is sent by asserting the EOI-line of the GPIB.
- The ^ character stands for 'concurrent with'.

Most controller programming languages send these terminators automatically, but allow changing it. So make sure that the terminator is one of those listed above.

Example: A terminated program message, consisting of 2 separate program message units and a terminator. The program message units in this example consist of a header, a header separator and a parameter:



Terminated Program Message

Basically there are two types of program headers:

- Compound headers.

  SCPI commands have a compound header consisting of one or more key words (mnemonics), mutually separated by the colon character (:). Such a colon may also precede the header.

- Common headers.

  The program messages that are standardized by IEEE-488.2 are called common commands. Their headers always start with the asterisk (*) character, e.g. *RST.

Each keyword in a compound command header represents a node in the command tree. The left most key word (SYSTem in the previous example) is the root node, representing the highest hierarchical level in the command tree. Subsequent key words represent sub nodes under the root node.

Long and Short Form

Mnemonics in SCPI program messages may be sent in either long or short form.

The long and short form of the mnemonics obey mainly the following rules:

- The long form consists of the full word but has a maximum of 12 characters.

  Example: INITIATE

- The short form consists of the first four characters of the long form.

  Example: INIT

  Exception: When the fourth character is a vowel and the four characters don't form the long form, then the short form is the first three characters only.
  Example: long form SWEEP, short form SWE
  But when the long form is FREE, then the short form is also FREE, because the short form is also the full word.

Sometimes a mnemonic is created from concatenated words or a phrase. In such a case the long form is the first character of each word, followed by the full length of the last word. The short form is created according to the same rules as mentioned before. Consider as an example: The long form mnemonic for Peak-To- Peak is PT-Peak. Its short form is PTP.

Either long or short form of a keyword may be used in a command. Any other form will cause a 'command' error.

Example: Either MAXIMUM (long form) or MAX (short form) may be used, but MAXI causes a 'command' error.

Notation Habit

In order to clarify the difference between short and long form, the short form in a syntax specification is shown in upper case. The remaining part of the long form is shown in lower case characters. Notice, however, that this does not specify the use of upper and lower case characters in the message that is actually sent. Bear in mind

that upper and lower case, as used in syntax specifications, is only a notation habit to facilitate distinction between long and short form.

Example: In the syntax specification DISPlay, the uppercase characters DISP specify the short form, whereas the full word DISPLAY specifies the long form.

Forgiving Listening

In program messages, either the long or short form may be used in any mix of upper or lower case characters. There is no semantic difference between upper and lower case in program messages. This instrument behavior is called forgiving or flexible listening. An application program may cause the following literal command to be sent over the bus:

dIsP:EnaB 1	short form used in a mix of upper and lower case.
Display:Enab 1	long and short form used in a mix of upper and lower case.

Numeric Suffixes

Instruments may consist of multiple identical components. For example, an instrument can have two input blocks. Selection of a particular component is done by adding numeric suffixes. A numeric suffix is a digit appended after the node that specifies the duplicated functionality. In the case of the instrument with the two input blocks, INPut1 and INPut2 provide the selection mechanism. The following rules are valid:

- Instruments that implement multiple capabilities, implement numeric suffix 1 as the default. If no suffix is appended to the node, suffix 1 is considered to be received.
- Instruments that do not implement multiple capabilities, may or may not accept a numeric suffix 1.

For example, the Philips PM 3585/90 Logic Analyzer has 12 groups of 8 channels. For each group the threshold value can be set. In order to set the threshold value of group 5 to 2.7 Volt, the following command can be sent:

:INPut:GROup5:THReshold 2.7

Command Tables

Command trees and command tables are commonly used to document an SCPI instrument command set. In this manual command tables are used. The following example shows a part of the command table of the STATUS subsystem:

HEADER	PARAMETERS	RESPONSE
:**STAT**us:**OPER**ation:**COND**ition?		<condition>
[:**EVEN**t]?		<events>
:**ENAB**le	<mask>	
:**ENAB**le?		<mask>
:**PRES**et		
:**QUEU**e[:**NEXT**]		<code>,<text>

The first column describes the supported program message headers, the second shows the mandatory and optional parameters while the third column illustrates the response messages for queries.

The command table shows the paths to be used for the syntax of a particular command within a subsystem. The mnemonic at the left top (root) of the command table (i.e. STATus) is the name of the subsystem. The syntax of a command header starts with this root node and continues until a leaf node.
For example, the command :STATus:OPERation:ENABle has STATus as root node and ENABle as leaf node. Leaf nodes are the last mnemonics in a command header. Each colon (:) in the command header moves the current path down one level from the root node in the command table.

Mnemonics enclosed between square brackets (e.g. [:EVENt]) indicate a default program message part, which can optionally be specified. This means that a program message may or may not contain the default mnemonic, without changing the meaning of the

message. For example, :STATus:QUEue:NEXT? is the same as :STATus:QUEue?

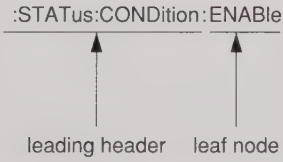
The following notes apply to commands:

- In general all commands have a query form. The response to a query is normally the parameter of its corresponding command.
- Commands that do not change a setting or a state of the instrument, have no query form. For example, commands that start an action (e.g. :INITiate to start the acquisition) have no corresponding query.
- Queries exist which do not have a command form. For example, the contents of a condition register (showing some aspects of the current state of the instrument) is only determined by the instrument itself and cannot therefore be programmed, but only queried.

Command Parsing

Although most SCPI commands and queries have a compound header (i.e. the header consists of more than one mnemonic), it is not always needed to repeat the full header path with every command. The full header consists of 2 parts, a leading header upto the leaf node and the leaf node itself.

Example:



Under certain conditions, the leading header may be omitted within a terminated program message. When a message is being interpreted, the instrument assumes that the command header starts with a root node after:

- A Program Message Terminator.

- A leading colon.

In all other cases, the first node (mnemonic) is assumed to be a node under the (leading) header path which is recived with the preceding command.

The following rules apply when a program message is sent to an instrument:

- The first message unit in a (terminated) program message must specify the complete header; i.e. from the root up to the leaf node. Because the instrument assumes in this case, that the command starts with a root node, a leading colon may or may not be sent.

  Example: :STATus:OPERation:CONDition?

- If a subsequent command does not start with a colon, the node is considered to be a leaf node under the header path of the preceding command (within the same program message).

  Example: :STATus:OPERation:CONDition?; ENABle?;

- If a subsequent command has a leading colon, the first node of the command is considered to be under the root.

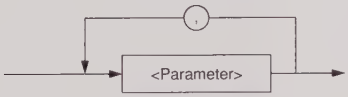
  Example: :STATus:OPERation:CONDition?;:STATus:PRESet;

- Common commands (i.e. commands preceded by *) do not effect the header path.

  Example: :STATus:OPERation:CONDition?;*ESR 30;ENABle?;

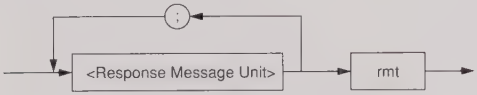
Syntax of Response Messages

The response of an SCPI instrument to a query is a response message unit, consisting of one or more parameters (data elements). Successive parameters are separated by a comma (,). There is no header returned, as the following syntax diagram shows:



Syntax of a Response Message Unit

If there are multiple queries in a program message, the multiple response message units are grouped together in the corresponding response message. Response message units are separated by a semicolon (;) and are terminated by a response message terminator (rmt) according to the following syntax:



Syntax of a terminated Response Message

The response message terminator (rmt) is NL^END, where:

NL^END is the ASCII <new line> code (equal to <line feed> code = 10 dec.) sent concurrently with the END message. The END message is sent by asserting the EOI-line of the GPIB.

An SCPI instrument will always send its response data in short form and in capitals. SCPI instruments do not send headers in their response messages, but parameters only. However, there are some exceptions, e.g. *LRN?. This query requires that the response can be sent back to the instrument. In these cases the header separator is a single space character (dec. 32).

Example of an instrument response:

1) An instrument is programmed with the following command:
   :FORMat:DATA OCTal
2) Next the following query is sent to the instrument:
   :FORMat:DATA?
3) The instrument will respond (precise talking) in short form and in capitals with:
   OCT

Parameters

SCPI defines parameters of several categories: numeric values, suffixes, booleans, expressions, decimal data, etc. These parameters are used in commands, queries and response messages.

Numeric Values

Numeric values (numerals) are used to represent decimal data. Numeric values may be presented in implicit (NR1) or explicit decimal point notation form (NR2) and may also contain exponential (base 10) notation (NR3). If a parameter may be given in each of the previous notation forms, it has the NRf (NR = NumeRic, f = flexible) format. Thus NRf means that either one of the following representation forms may be used:

<NR1> : Integer numeric (+ or - sign allowed).

Example: -179

<NR2> : Real numeric with decimal point (+ or - sign allowed).

Example: -179.56

<NR3> : Floating point numeric with decimal point, exponent (E) and mantissa (+ or - sign allowed).

Example: -1.7956E+02

Furthermore, several keywords are regarded as special forms of numeric data, such as: MINimum, MAXimum, etc. The keywords, which are allowed within a particular command or query, are always explicitly specified.

MINimum : This keyword represents the minimum value for a parameter.

MAXimum : This keyword represents the maximum value for a parameter.

Example: SYSTem:TIME? --> MAXimum,MINimum,MAXimum
<-- 23,0,59

Non-decimal Data

Non-decimal data allows the specification of numeric data in bases other than 10. These specifications start with a # character followed by the base character (B for Binary, Q for Octal, H for Hexadecimal) followed by one or more digits in the appropriate base.

Examples : #B10 = Binary 10 = Decimal 2

#Q10 = Octal 10 = Decimal 8

#H10 = Hexadecimal 10 = Decimal 16

Booleans

A Boolean parameter is used to specify a single binary condition that is either true or false. The allowed values of a boolean parameter are:

ON or 1 : means condition true

OFF or 0 : means condition false

For example, the command DISPlay:ENABle 1 switches the display on.

Booleans in a response to a query are always represented as 0 or 1; i.e. in this case the keywords ON and OFF are not used.

Block Data

Block data is used to transfer any 8-bit coded data. The data starts with a preamble that specifies the format and length of the block data element.
The following block formats are defined:

- Indefinite length block.

  This block begins with the header #0, followed by any number of data bytes.
  This block must be terminated by a program or response message terminator in order to indicate the end of the block.

- Definite length block.

  This block begins with #, followed by one ASCII character specifying the number of digits in the length field, followed by the length field itself indicating the number of data bytes, followed by the data bytes itself.

  Example: #212<db1><db2><db3>.....<db11><db12>

Character Data

Character data is used to specify keywords. The same rules are valid as for the mnemonics being used in command headers.

Examples: POSitive, NEGative, EITHer, etc.

String Data

String data is used to specify a sequence of ASCII characters. A string data element is enclosed between single (') or double (") quotes. Response data strings are always enclosed between double quotes.
Double quotes should be used if a single quote is to be specified as a string character.

Example: "This is a string"

Expression Data

Expression program data elements are enclosed in parenthesis (). Two types of expression data are defined:

- Numeric expressions

  Example: (215MOD16)

- Channel lists

  Example: (@1,3,4:20,22:29,33)

Suffixes

SCPI defines several suffixes which are used to express a unit or multiplier being associated with decimal numeric data in commands and queries. For example, mV for milli Volt.
Response messages do not have a suffix. The returned value is to be associated with standard units, such as V, A, Hz, etc.

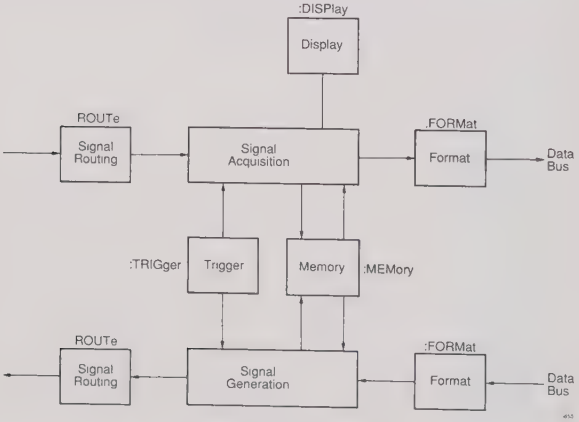
Chapter 3
The Instrument Model

General Instrument Model 3-2
Subsystems 3-3
Analyzer Instrument Model 3-6

From a users point of view, programming an instrument from a remote site differs completely from local operation. In order to facilitate this way of programming and to achieve instrument compatibility at various levels, SCPI uses a generalized instrument model.

General Instrument Model

SCPI defines a general model for programmable instruments. The following diagram shows how SCPI looks at an instrument and how the functionality of such an instrument is presented:



General Instrument Model

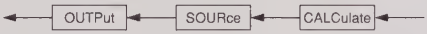
This model covers three basic types of an instrument:

- Sense instruments for signal acquisition. For example, a Logic Analyzer.
- Source instruments for signal generation. For example, a pulse generator.
- Switch instruments for signal routing. For example, a multiplexer.

SCPI further partitions the "signal acquisition" function of sense instruments into the functional blocks INPUT, SENSE and CALCULATE, as shown in the following diagram:

Signal Acquisition

A similar partitioning is defined for the "signal generation" function of source instruments into OUTPUT, SOURCE and CALCULATE subsystems:



Signal Generation

These diagrams show the flow of measurement and signal data through the instrument. They enable the functions of an instrument to be classified into different groups. The hierarchical command structure of SCPI is based upon this partitioning.

Subsystems

To be able to program the functions of an instrument, it is necessary to understand the instrument model.

The SCPI commands and queries are divided into groups, called subsystems. The control of the functions present in a function block, is done via the commands in the related subsystem. The commands to control a particular subsystem begin with the name of that subsystem. For instance, stored acquisition data is associated with the MEMORY subsystem.
Commands to access this data have :MEMory as leading (root) node in the command header.

Subsystems of an instrument, which are directly related to signal processing, are:

CALCULATE: This subsystem performs the post-processing functions that are provided on the acquired data. It is logically placed between the SENSE subsystem and data output functions, such as the DISPLAY and FORMAT subsystem.

DISPLAY: This subsystem contains the functions associated with the presentation of data. Presentation can be in any form, such as textual or graphical. The data being presented can be measurement data, user interactions, status data, etc.

FORMAT: This subsystem deals with conversion from one data format into an other. This function is especially used for transferring data over a remote interface like the GPIB or RS-232.

INPUT: This subsystem controls the characteristics of a sensor input, such as impedance. These are typical signal conditioners, before the signal is processed by the SENSE subsystem.

OUTPUT: This subsystem controls the characteristics of a source output port, such as attenuation.

ROUTE: This subsystem is basically a switch function that routes signals to and from the instrument. For SENSE instruments, the signal is routed from the instrument's input ports to the INPUT block. For SOURCE instruments the signal is routed to the output ports.

SENSE: This subsystem contain the functions which perform the real data acquisition. Its purpose is to convert the actual signal into internal data.

TRIGGER: This subsystem controls the data acquisition (or data generation) by synchronizing the instrument actions with internal and/or external events.

Other subsystems, not directly related to signal processing, are:

MEMORY: This subsystem provides the functions which operate on the instrument memory.

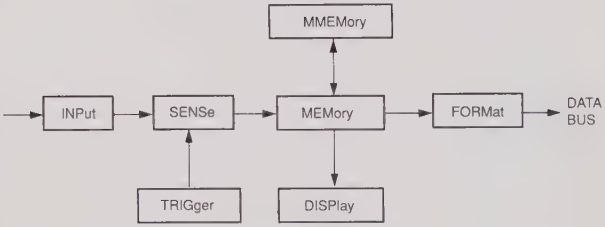
MMEMORY: This subsystem contains the functions which provide access to mass memory.

STATUS: This subsystem controls the SCPI status reporting system which includes status registers, event registers, event transition filters, etc.

SYSTEM: This subsystem includes the more general functions of the instrument, which are not directly related to the purpose of the instrument.

Analyzer Instrument Model

The following diagram shows the instrument model of the PM 3580/ PM 3585 Logic Analyzer:



Analyzer Instrument Model

Subsystems for which (currently) no commands are provided, are not shown in the diagram. Other subsystems, not shown but for which commands are available, are SYSTEM and STATUS.

Chapter 4
Command Descriptions

Introduction 4-4
- Syntax Notation 4-5
- Standard Syntactical Items 4-6
- Other Syntactical Items 4-8

DISPlay Subsystem 4-9
- :DISPlay:ENABle 4-11
- :DISPlay:INPut 4-13
- :DISPlay:MESSage 4-15
- :DISPlay:QUEStion 4-17

FORMat Subsystem 4-19
- :FORMat:DATA 4-21
- :FORMat:SREGister 4-23
- :FORMat:STAMp 4-25

INPut Subsystem 4-27
- :INPut:GROup:THReshold 4-29

MEMory Subsystem 4-33
- :MEMory:CLEar:MEASurement 4-35
- :MEMory:COPY:MEASurement 4-36
- :MEMory:DUMP:MEASurement 4-37
- :MEMory:EXCHange:MEASurement 4-39
- :MEMory:TABLe:ACCess:FORWard 4-40
- :MEMory:TABLe:CHANlist 4-42
- :MEMory:TABLe:COMPare:SKEW 4-43
- :MEMory:TABLe:POINts 4-46
- :MEMory:TABLe:POSition:NUMBer 4-47
- :MEMory:TABLe:POSition:POINt 4-49
- :MEMory:TABLe:POSition:STAMp 4-51
- :MEMory:TABLe:POSition:VALue 4-53
- :MEMory:TABLe:SAMPle:DATA 4-56

:MEMory:TABLe:SAMPle:DURation 4-59
:MEMory:TABLe:SAMPle:STAMp 4-61
:MEMory:TABLe:SAMPle:STATus 4-63
:MEMory:TABLe:SAMPle:VALue 4-65
:MEMory:TABLe:SAMPle:SELect 4-67
MMEMory Subsystem 4-69
:MMEMory:CATalogue 4-71
:MMEMory:COPY 4-73
:MMEMory:DELete 4-74
:MMEMory:INITialize 4-75
:MMEMory:LOAD:CONFiguration 4-76
:MMEMory:LOAD:MEASurement 4-77
:MMEMory:LOAD:SYMBols 4-79
:MMEMory:LOAD:SYSTem 4-80
:MMEMory:MOVE 4-81
:MMEMory:STORe:MEASurement 4-82
:MMEMory:STORe:SCReen 4-84
:MMEMory:TRANsfer 4-85
SENSe Subsystem 4-87
:SENSe:CHANnel:ACTivity 4-89
STATus Subsystem 4-91
*CLS 4-93
*ESE 4-94
*ESR 4-96
*SRE 4-97
*STB 4-98
:STATus:OPERation:CONDition 4-99
:STATus:OPERation:ENABle 4-100
:STATus:OPERation:EVENt 4-101
:STATus:OPERation:NTRansition 4-102
:STATus:OPERation:PTRansition 4-103
:STATus:PRESet 4-104
:STATus:QUEStionable:CONDition 4-105
:STATus:QUEStionable:ENABle 4-106
:STATus:QUEStionable:EVENt 4-107
:STATus:QUEStionable:NTRansition 4-108
:STATus:QUEStionable:PTRansition 4-109
:STATus:QUEue:NEXT 4-110
SYNChronization Subsystem 4-111
*OPC 4-113
*WAI 4-114

SYSTem Subsystem 4-115
*IDN 4-117
*OPT 6-118
*RST 4-119
:SYSTem:BOOT 4-120
:SYSTem:COMMunicate:GPIB:ADDRess 4-121
:SYSTem:DATE 4-123
:SYSTem:ERRor 4-125
:SYSTem:KEY 4-126
:SYSTem:KLOck 4-128
:SYSTem:PRESet 4-129
:SYSTem:TIME 4-130
:SYSTem:VERSion 4-132
TEST Subsystem 4-133
*TST 4-135
TRIGger Subsystem 4-137
:ABORt 4-139
:INITiate 4-140
:STOP 4-141

COMMAND DESCRIPTIONS

Introduction

This chapter describes all commands and queries supported by the PM 3580/PM 3585 logic analyzer. These commands and queries are grouped together into the functional subsystems defined by the SCPI standard 1992.0. Note that not all commands/queries are SCPI compatible. This means that some of them are not SCPI confirmed and/or not approved (yet).

Each subsystem starts with a command table showing which commands and queries are supported for that subsystem.

The description of a command and its corresponding query is combined. Each command/query description starts on a new page and consists, if applicable, of the following parts:

Command:	Describes the syntax of the command including the mandatory and optional parameters.
Query:	Describes the syntax of the query form, including the parameters.
Response:	Shows the syntax of the response to the query.
Description:	Gives a short explanation about the command, query and response.
See also:	Gives reference(s) to other chapter(s) in this manual, and other documents where additional information can be found about the command/ query.
Examples:	One or more examples of how the command/ query can be used.
***RST condition:**	The effect of a *RST command on the described command/query; i.e. the default setting.
Errors:	Describes the command/query specific error conditions. A list of all possible errors is given in chapter 6: Error and Event Reporting.

COMMAND DESCRIPTIONS

Syntax Notation

The description of the commands and queries follows the syntax notation rules as described in SCPI 1992 Volume 1 Section 5.

The main rules are:

- ::= means "is defined as".
- Anything enclosed in angle brackets, **<** and **>**, is a non-terminal syntactical item.
- Square brackets, **[** and **]**, enclose a construct that is optional; i.e. a construct that may occur zero or once only.

  For example: [<key>] means that the syntactical item <key> may be present or may be omitted.

- Braces, **{** and **}**, enclose a construct that may occur zero or more times.

  For example: {<key>} represents an empty sequence or a sequence that consists of one or more syntactical items <key>.

  <key>{<key>} represents a sequence that consists of at least one syntactical item.

  [<key>{,<key>}] represents a sequence that may be omitted or a sequence that consists of one item <key> or a sequence of more than one item <key> separated by a comma.

- The vertical bar, **|**, separates alternatives.

  For example: <digit> | <letter> means that either the syntactical item <digit> or <letter> may be specified.

COMMAND DESCRIPTIONS

Standard Syntactical Items

In the description of the commands and queries the following standard syntactical items are used:

<Boolean> ::= 0 | OFF | 1 | ON

As parameter, <Boolean> is used to specify a single binary condition that is either false (0 or OFF) or true (1 or ON).

As response to a query, <Boolean> represents the current state which is either false (0) or true (1).

See also: SCPI 1992 Volume 1 Section 7.3

<chan_list> ::= (@<chan_range>{,<chan_range>})

<chan_range> ::= <chan_nr>[:<chan_nr>]

<chan_nr> ::= <NR1>

Range 1 to 96 inclusive.

<chan_list> is used to specify one or more channel numbers.

Example: (@1,3,4:20)

Specifies the channels 1, 3 and 4 up to and including 20.

See also: SCPI 1992 Volume 1 Section 8.3.2

<character_data> ::= A sequence of one or more characters.

For this data type, the same rules are valid as for the mnemonics used in command headers.

Examples: MINimum, TRIGger, etc.

COMMAND DESCRIPTIONS

See also: IEEE-488.2 1987 Sections 7.7.1 and 8.7.1

<dec_data> ::= <NRf>

<non_dec_data> ::= <bin_value> | <oct_value> | <hex_value>

Allows specification of numeric data in bases other than 10.

See: IEEE-488.2 1987 Section 7.7.4.

<NR1> ::= Integer numeric value.

See: IEEE-488.2 1987 Section 7.7.2

<NR2> ::= Real numeric value.

See: IEEE-488.2 1987 Section 7.7.2.

<NR3> ::= Floating point numeric value.

See: IEEE-488.2 1987 Section 7.7.2.

<NRf> ::= <NR1> | <NR2> | <NR3>

<numeric_data> ::= <dec_data> | <non_dec_data>

<string_data> ::= Data enclosed between single (') or double (") quotes.

<string_data> in query responses is always enclosed between double quotes.

Example: "This is a string"

COMMAND DESCRIPTIONS

Other Syntactical Items

In addition to the standard syntactical items described in the previous section, the following non-standard items are used:

<coor> ::= <symb_coor> | <num_coor>

<symb_coor> ::= CENTer

<num_coor> ::= <numeric_data>

Used to specify a symbolic or absolute screen coordinate (x or y position).

<file_name> ::= <string_data>

Used to specify the name of a file on diskette. It follows the DOS filename convention; i.e. it consists of a name optionally followed by an extension. The extension, if present, is separated from the name by a period.

The name may consist of up to eight characters. The extension may contain no more than three characters.

Examples: "SYSTEM" and "MEAS.AUT"

<stamp> ::= <numeric_data> | <block>

Used to specify an acquisition timestamp in units of 5ns.

DISPlay Subsystem

:DISPlay

HEADER	PARAMETERS	RESPONSE
:**DISP**lay:**ENAB**le	<state>	
→ :**ENAB**le?		<state>
→ :**INP**ut?	<x>,<y>,<title>,<text>,<lth>[,<input>]	<input>
→ :**MESS**age	<x>,<y>,<title>,<message>	
→ :**QUES**tion?	<x>,<y>,<title>,<question>, <def_answer>, <answer>,<answer>[,answer>]	<answer_nr>

:DISPLAY:ENABLE

Command

:DISPlay:ENABle <state>

<state> ::= <Boolean>

Query

:DISPlay:ENABle?

Response

<state>

<state> ::= 0 | 1

Description

The DISPlay:ENABle command controls whether the display should be visible (ON, 1) or should be switched off (OFF, 0).

The query form of this command returns the current display state (1 or 0).

When the display is switched off, updates to the screen are still made in screen memory. So when the display is switched on again, the correct information will be shown.

See also

SCPI 1992 Volume 2 Section 7.5

:DISPLAY:ENABLE

Examples

→ :DISPlay:ENABle OFF

→ :DISPlay:ENABle?
← 0

→ :DISPlay:ENABle 1

→ :DISPlay:ENABle?
← 1

**RST condition*

1 (ON)

Errors

-224, "Illegal parameter value"

The specified state (<state>) is not a <Boolean> value.

:DISPLAY:INPUT

Query

:DISPlay:INPut? <x>,<y>,<title>,<text>,<lth>[,<input>]

<x> ::= <coor>
Range 0 to 720 inclusive or CENTer.

<y> ::= <coor>
Range 0 to 480 inclusive or CENTer.

<title> ::= <string_data>
Maximum of 60 characters.

<text> ::= <string_data>
Maximum of 700 characters.

<lth> ::= <numeric_data>
Range 1 to 60.

<input> ::= <string_data>
Maximum of 60 characters.

Response

<input>

<input> ::= <string_data>

Description

The :DISPlay:INPut query enables the controller to read a free format text string from the local operator.

The dialogue between the analyzer and the local operator is done via the analyzer keyboard and a popup window on the analyzer display. <x> and <y> define the coordinates of the top left of the popup and <title> specifies the popup title. <text> is displayed in the popup window and can be used by the controller to explain what type of information is required. <lth> defines the maximum length of the input string and <input> is the initial input.

:DISPLAY:INPUT

See also

:DISPlay:MESSage command

:DISPlay:QUEStion? query

Examples

With the following query the controller asks the local operator to enter the serial number of the unit under test. The popup window should be centered in both X (horizontal) and Y (vertical) direction. "SERIAL_NR" is the title of the popup and no additional information about what should be entered is required (""). The maximum length of the input string is 8 characters and the initial string is "SN.....".

→ :DISPlay:INPut? CENTer,CENTer,"SERIAL_NR","",8,"SN....."

← "SN248124"

**RST condition*

Not applicable

Errors

-222,"Data out of range"

Coordinates (<x>, <y>) or input length (<lth>) are not within valid range.

-223,"Too much data"

Specified title exceeds 60 characters text exceeds 700 characters or default input exceeds 60 characters.

:DISPLAY:MESSAGE

Command

:DISPlay:MESSage <x>,<y>,<title>,<message>

<x> ::= <coor>
Range 0 to 720 inclusive or CENter.

<y> ::= <coor>
Range 0 to 480 inclusive or CENTer.

<title> ::= <string_data>
maximum of 60 characters.

<message> ::= <string_data>
maximum of 700 characters.

Description

The :DISPlay:MESSage command enables the controller to send a message to the local operator.

The message is displayed on the analyzer screen in the form of a popup window. <x> and <y> define the coordinates of the left top of the window, <title> specifies the title of the popup and <message> is the text string displayed in the popup.

See also

:DISPlay:INPut? query

:DISPlay:QUEStion? query

Examples

In this example the controller sends the message "I am ready" to the local operator. The lefthand side of the popup is at screen coordinate 100 and the popup is centered in the Y (vertical) direction. The popup title is "MESSAGE".

→ :DISPlay:MESSage 100,CENTer,"MESSAGE","I am ready"

:DISPLAY:MESSAGE

**RST condition*

Not applicable.

Errors

- 222, "Data out of range"

  Popup coordinates (<x>, <y>) not within allowed range.

- 223, "Too much data"

  Specified title (<title>) or message (<message>) longer than allowed.

:DISPLAY:QUESTION

Query

:DISPLAY:QUEStion? <x>,<y>,<title>,<question>,
<def_answer>,<answer>,
<answer>[,<answer>]

<x> ::= <coor>
Range 0 to 720 inclusive or CENTer.

<y> ::= <coor>
Range 0 to 480 inclusive or CENTer.

<title> ::= <string_data>
Maximum of 60 characters.

<question> ::= <string_data>
Maximum of 700 characters

<def_answer> ::= <numeric_data>
Range 1 to 5.

<answer> ::= <string_data>
Maximum of 70 characters.

Response

<answer_nr>

<answer_nr> ::= <NR1>
Range 1 to 5 inclusive.

Description

The :DISPlay:QUEStion query enables the controller to send a question to the local operator who should select an answer from a number of predefined answers. The answer number is returned to the controller.

:DISPLAY:QUESTION

The question and possible answers are presented to the local operator in the form of a popup window. <x> and <y> specify the coordinates of the left top of the popup and <title> is the popup title.

See also

:DISPlay:INPut? query

:DISPlay:MESSage command

Examples

By sending the following query the controller asks the local operator if it may continue. The left top of the popup should be placed at position (100,200) of the analyzer screen. The default answer is "Yes" (answer number 1).

In this example the local operator has selected answer number 2 (i.e. "No"), via the analyzer's front panel, which is returned to the controller.

→ :DISPlay:QUEStion? 100,200,"","Continue?",1,"Yes","No"

← 2

**RST condition*

Not applicable.

Errors

-222, "Data out of range"

Popup coordinates (<x>, <y>) or default answer number (<def-answer>) not within allowed range.

-223, "Too much data"

The length of the popup title (<tiltle>), question (<question>) or any of the answers (<answer>) is more than allowed.

FORMat Subsystem

:FORMat

HEADER	PARAMETERS	RESPONSE
:**FORM**at[:**DATA**]	<type>[,<length>]	
→ [:**DATA**]?		<type>[,<length>]
→ :**SREG**ister	<type>	
→ :**SREG**ister?		<type>
→ :**STAM**p	<type>[,<length>]	
→ :**STAM**p?		<type>[,<length>]

:FORMAT:DATA

Command

:FORMat:DATA <type>[,<length>]

<type> ::= ASCii | BINary | HEXadecimal | OCTal | PACKed

<length> ::= <numeric_data>

Range 0 to ....
Default: 0

Query

:FORMat:DATA?

Response

<type>[,<length>]

<type> ::= ASC | BIN | HEX | OCT | PACK

<length> ::= <NR1>

Range 1 to ...

Description

The :FORMat:DATA command selects the format type (<type>) and length (<length>) of measurement data in query responses.

The length defines the minimum number of digits returned excluding the BINary (#B), HEXadecimal (#H), Octal (#Q) and PACKed (#xxx) prefixes. A specification of zero length is treated as if the <length> parameter has been omitted.

The :FORMat:DATA? query shows the currently selected format type and length.

The <length> is not returned if this parameter has been omitted (or specified as 0) in the previous FORMat:DATA command.

:FORMAT:DATA

See also

SCPI 1992 Volume 2 Section 8.2

Examples

→ :FORMat:DATA BINary,12

→ :FORMat:DATA?
← BIN,12

→ :FORMat:DATA OCTal,0
← OCT

**RST condition*

Type ASCii and default length (0).

:FORMAT:SREGISTER

Command

:FORMat:SREGister <type>

<type> ::= ASCii | BINary | HEXadecimal | OCTal

Query

:FORMat:SREGister?

Response

<type>

<type> ::= ASC | BIN | HEX | OCT

The :FORMat:SREGister command selects the type (<type>) of the response to queries for:

- CONDITION, EVENT and ENABLE registers.
- NTRANSITION and PTRANSITION filters.
- IEEE-488.2 status registers.

The length of the response can not be specified. This length is implicitly derived from the selected response type and the size (in bits) of the required register. For example if BINary is being selected, the response to *STB? will consist of #B followed by 8 binary digits.

The :FORMat:SREGister? query shows the currently selected format type.

See also

SCPI 1992 Volume 2 Section 8.4

:FORMAT:SREGISTER

Examples

→ :FORMat:SREGister HEXadecimal

→ :FORMat:SREGister?
← HEX

→ *STB?
← #H27

**RST condition*

ASCii

:FORMAT:STAMP

Command

:FORMat:STAMp <type>[,<length>]

<type> ::= ASCii | BINary | HEXadecimal | OCTal | PACKed
<length> ::= <numeric_data>
Range 0 to ....
Default: 0

Query

:FORMat:STAMp?

Response

<type>[,<length>]

<type> ::= ASC | BIN | HEX | OCT | PACK
<length> ::= <NR1>
Range 1 to ....

Description

The :FORMat:STAMp command selects the format type (<type>) and length (<length>) of timestamps in query responses.

The length defines the minimum number of digits returned excluding the BINary (#B), HEXadecimal (#H), Octal (#Q) and PACKed (#xxx) prefixes. A specification of zero length is treated as if the <length> parameter has been omitted.

The :FORMat:STAMp? query returns the currently selected format type and length. The <length> is not returned if it has been omitted in the previous FORMat:STAMp command (or has been specified as 0).

:FORMAT:STAMP

See also

SCPI 1992 Volume 2 Section 8.2.

Examples

→ :FORMat:STAMp PACKed,8
→ :FORMat:STAMp?
← PACK,8

→ :FORMat:STAMp ASCii
→ :FORMat:STAMp?
← ASC

**RST condition*

Type ASCii and default length (0).

INPut Subsystem

:INPut

HEADER	PARAMETERS	RESPONSE
:**INP**ut:**GRO**up[<goup_nr>][:**THR**eshold]		<threshold>
→ [:**THR**eshold]?	[<threshold>]	<threshold>

:INPUT:GROUP:THRESHOLD

Command

:INPut:GROup[<group_nr>][:THReshold] <threshold>

<group_nr> ::= <NR1>
Range 1 to 12 inclusive
Default: 1

<threshold> ::= MINimum | MAXimum | TTL | ECL | <value>

<value> ::= <numeric_data>
Range -3.0 to 12.0 inclusive.

Query

:INPut:GROup[<group_nr>][:THReshold]? [<threshold>]

<group_nr> ::= <NR1>
Range 1 to 12 inclusive.
Default: 1

<threshold> ::= MINimum | MAXimum | TTL | ECL

Response

<threshold>

<threshold> ::= TTL | ECL | <value>

<value> ::= <NR2>
Range -3.0 to 12.0 inclusive

:INPUT:GROUP:THRESHOLD

Description

The :INPut:GROup:THReshold command sets the threshold of the specified group (<group_nr>) to the given value (<threshold>). The execution of this command is suspended if a local dialoque or the acquisition is in progress.

The :INPut:GROup:THReshold? query returns the threshold of the specified group (<group_nr>) or, if a parameter is given, the <NR2> value that corresponds with that parameter.

The analyzer channels are divided into groups of eight channels each. For each group the threshold value can be set. The relation between the group number and the Pod/Channels is shown in the following table:

Group Nr	POD	Channels
1	1	0-7
2	1	8-15
3	2	0-7
4	2	8-15
5	3	0-7
6	3	8-15
7	4	0-7
8	4	8-15
9	5	0-7
10	5	8-15
11	6	0-7
12	6	8-15

See also

Reference guide: FORMAT Menu-Thresholds

:INPUT:GROUP:THRESHOLD

Examples

→ :INPut:GROup7:THReshold -2.7

→ :INPut:GROup7:THReshold?
← -2.7

→ :INPut:GROup:THReshold? TTL
← 1.4

→ :INPut:GROup3:THReshold? MAX
← 12.0

**RST condition*

At *RST the threshold values for all groups are set to TTL.

Errors

-131,"Invalid suffix"

<group_nr> outside range of 1 ... 12

-222,"Data out of range"

Threshold value outside range of -3.0 to 12.0

-241,"Hardware missing"

Specified group number not present in hardware configuration for this analyzer.

:INPUT:GROUP:THRESHOLD

MEMory Subsystem

:MEMory

HEADER	PARAMETERS	RESPONSE
:**MEM**ory:**CLE**ar:**MEAS**urement	[<meas_name>]	
:**COPY**:**MEAS**urement	<src>[,<dst>]	
:**DUMP**:**MEAS**urement?	[<meas_name>]	{<block>}
:**EXCH**ange:**MEAS**urement		
:**TABL**e[:**ACC**ess]:**FORW**ard	<state>	
:**FORW**ard?		<state>
:**CHAN**list?		<chan_list>
:**COMP**are[<anal_nr>] :**SKEW**	<skew_value>	
:**SKEW**?		[<skew_value>]
:**POIN**ts?		<points>
:**POS**ition:**NUMB**er	<position>	
:**NUMB**er?		<sample_nr>
[:**POIN**t]	<position>	
[:**POIN**t]?		<point-nr>
:**STAM**p	<position>	
:**STAM**p?		<stamp>
:**VAL**ue	<value>[,<chan_spec>]	
:**VAL**ue?	<value>[,<chan_spec>]	<point_nr>
:**SAMP**le:**DATA**?	[<count>[,<chanspec>]]	{<acq_data>}
:**DUR**ation?	[<count>]	{<duration>}
:**STAM**p?	[<count>]	{<stamp>}
:**STAT**us?	[<count>]	{<status>}
:**VAL**ue?	[<count>[,<chan_spec>]]	{<value>}
:**SEL**ect	<table_name>	
:**SEL**ect?		<table_name>

:MEMORY:CLEAR:MEASUREMENT

Command

:MEMory:CLEar:MEASurement [<meas_name>]

<meas_name> ::= NEW | REFerence
Default : NEW

Description

The :MEMory:CLEar:MEASurement command erases the contents of the measurement specified by <meas_name>.
The execution of this command is suspended if a local dialoque is in progress.

See also

Chapter 10: Acquisition Data Access.

Examples

→ :MEMory:CLEar:MEASurement

→ :MEMory:CLEar:MEASurement REFerence

:MEMORY:COPY:MEASUREMENT

Command

:MEMory:COPY:MEASurement <src_name> [, <dst_name>]

<src_name> ::= NEW | REFerence

<dst_name> ::= NEW | REFerence

Default: NEW if <src_name> is REFerence.
REFerence if <src_name> is NEW.

Description

The :MEMory:COPY:MEASurement command copies the contents of the measurement specified by <src_name> to that specified by <dst_name>.
The execution of this command is suspended if a local dialoque is in progress.

See also

Reference Guide: Display Special Functions Menu

Chapter 10: Acquisition Data Access.

Examples

Each of the following commands copies the contents of NEW to the REFerence measurement structure:

→ :MEMory:COPY:MEASurement NEW, REFerence
→ :MEMory:COPY:MEASurement NEW

The following command copies REFerence to NEW:

→ :MEMory:COPY:MEASurement REFerence

:MEMORY:DUMP:MEASUREMENT

Query

:MEMory:DUMP:MEASurement? [<meas_name>]

<meas_name> ::= NEW | REFerence
default : NEW

Response

<block>{,<block>}

<block> ::= Definite length arbitrary block.
See IEEE-488.2 1987 Section 8.7.9

Description

The :MEMory:DUMP:MEASurement? query returns a measurement header followed by the contents of the measurement specified by <meas_name> (or by default NEW) to the controller in N definite length arbitrary blocks.

The length of the actual data contained in each block, with the exception of the last one, is 65024 bytes. The length of the data contained in the last block varies between 1 and 65024 bytes.

The format of the actual data transferred (concatenation of the data contained in the blocks after removal of the block headers (#...)) is in accordance with the PM 3580/PM 3585 measurement file format.

If data is requested from NEW and the acquisition is running (not in IDLE state), this query implicitly waits for acquisition completion.

See also

Chapter 10: Acquisition Data Access.

User Manual Chapter 10: Measurement File Format.

:MEMORY:DUMP:MEASUREMENT

Examples

→ :MEMory:DUMP:MEASurement?
← <block>,...,<block>

→ :MEMory:DUMP:MEASurement? REFerence
← <block>,...,<block>

**RST condition*

Not applicable.

Errors

-230, "Data corrupt or stale"
Specified measurement is empty.

:MEMORY:EXCHANGE:MEASUREMENT

Command

:MEMory:EXCHange:MEASurement

Description

The :MEMory:EXCHange:MEASurement command exchanges the contents of the NEW and REFerence memory storages.
The execution of this command is suspended if a local dialoque is in progress.

See also

Reference guide: Display Special Functions Menu.

Chapter 10: Acquisition Data Access.

Examples

→ :MEMory:EXCHange:MEASurement

:MEMORY:TABLE:ACCESS:FORWARD

Command

:MEMory:TABLe[:ACCess]:FORWard <state>

<state> ::= <Boolean>

Query

:MEMory:TABLe[:ACCess]:FORWard?

Response

<state>

<state> ::= 0 | 1

Description

The :MEMory:TABLe:ACCess:FORWard command controls the direction in which data is read (via :MEMory:TABLe:SAMPle queries) from the memory tables; i.e. forward (ON or 1) direction (increasing point numbers) or reverse (OFF or 0) direction (decreasing point numbers).

The :MEMory:TABLe:ACCess:FORWard query returns the current direction; i.e. 0 (reverse) or 1 (forward).

Examples

→ :MEMory:TABLe:ACCess:FORWard ON
→ :MEMory:TABLe:ACCess:FORWard?
← 1

**RST condition*

1 (i.e. forward access)

:MEMORY:TABLE:ACCESS:FORWARD

Errors

-224,"Illegal parameter value"

The specified direction (<state>) is not a legal <Boolean> value.

:MEMORY:TABLE:CHANLIST

Query

:MEMory:TABLe:CHANlist?

Response

<chan_list>

<chan_list> ::= See SCPI 1992 Volume 1 Section 8.3.2

Description

The :MEMory:TABLe:CHANlist? query returns a channel list indicating for which channels data is stored in the currently selected memory table.

See also

Chapter 10: Acquisition Data Access

Examples

→ :MEMory:TABLe:CHANlist?
← (@32:23,18,16:14,1)

**RST condition*

Not applicable.

:MEMORY:TABLE:COMPARE:SKEW

Command

:MEMory:TABLe:COMPare[<anal_nr>]:SKEW <skew_value>

<anal_nr> ::= <NR1>
1 or 2. Default 1.

<skew_value> ::= <symb_value> | <num_value>

<symb_value> ::= MINimum | MAXimum

<num_value> ::= <numeric_value>
Skew value in units of 5ns. Range 0 to 13107 inclusive.

Query

:MEMory:TABLe:COMPare[<anal_nr>]:SKEW? [<symb_value>]

<anal_nr> ::= <NR1>
1 or 2. Default 1.

<symb_value> ::= MINimum | MAXimum

Response

<num_value>

<num_value> ::= <NR1>

Description

The :MEMory:TABLe:COMPare:SKEW command defines the skew for the analyzer specified by <anal_nr>.

:MEMORY:TABLE:COMPARE:SKEW

The skew value defines the maximum acceptable shift in time (specified in units of 5ns) between any transition in the NEW and its corresponding transition in the REFerence measurement when timing acquisition data is searched for differences or equalities.

Corresponding transitions with a distance in time of no more than the specified skew value are considered to match.

The query form of this command returns the current skew value or, if a parameter has been specified, the corresponding skew value.

See also

:MEMory:TABle:POSition:VALue command.

Chapter 10: Acquisition Data Access.

Examples

→ :MEMory:TABLe:COMPare:SKEW 10

→ :MEMory:TABLe:COMPareSKEW?
← 10

→ :MEMory:TABLe:COMPare:SKEW? MAXimum
← 13107

Errors

-131,"Invalid suffix"

<anal_nr> not 1 or 2

-222,"Data out of range"

Skew value outside range of 0 to 13107

:MEMORY:TABLE:COMPARE:SKEW

**RST condition*

At *RST the skew value is set to 1 (i.e. 5ns) for the PM 3585 and set to 2 (i.e. 10ns) for the PM 3580 model.

:MEMORY:TABLE:POINTS

Query

:MEMory:TABLe:POINts?

Response

<points>

<points> ::= <NR1>
Range 0 to .....

Description

The :MEMory:TABLe:POINts? query returns the number of points in the currently selected memory table.

If a table has not yet been selected, the analyzer first selects a table according the algorithm described in Chapter 10: Acquisition Data Access.
The execution of this query is suspended if it refers to the NEW measurement and an acquisition is in progress.

The query :MEMory:TABLe:POINts? returns the same result as the query :MEMory:TABLe:POSition? MAXimum minus 1.

See also

Chapter 10: Acquisition Data Access.
SCPI 1992: Volume 2 Section 11.6.3.2.1

Examples

→ :MEMory:TABLe:POINts?
← 2048

:MEMORY:TABLE:POSITION:NUMBER

Command

:MEMory:TABLe:POSition:NUMBer <position>

<position> ::= <sample_nr> | <symb_pos>
<sample_nr> ::= <NR1>
<symb_pos> ::= MINimum | MAXimum | TRIGger

Query

:MEMory:TABLe:POSition:NUMBer? [<symb_pos>]

<symb_pos> ::= MINimum | MAXimum | TRIGger

Response

<sample_nr>

<sample_nr> ::= <NR1>

Description

The :MEMory:TABLe:POSition:NUMBer command sets the current position in the memory table to the point (entry) number which corresponds with the specified sample number (<sample_nr>) or symbolic position (<symb_pos>).

The query form of this command returns the sample number which corresponds with specified symbolic position or, if omitted, the sample number which corresponds with the current point (entry) number in the table.

See also

Chapter 10: Acquisition Data Access.

:MEMORY:TABLE:POSITION:NUMBER

Examples

Request for the sample number of the trigger point:

→ :MEMory:TABle:POSition:NUMBer? TRIGger
← 0

Set table access position on the first sample (i.e. the first table entry) and ask for the current sample number (i.e. the number of samples before the trigger point):

→ :MEMory:TABLe:POSition:NUMBer MINimum
→ :MEMory:TABLe:POSition:NUMBer?
← -23

Obtain the sample number of the last entry in the table (i.e. the number of samples after the trigger point):

→ :MEMory:TABLe:POSition:NUMBer? MAXimum
← 976

**RST condition*

Position is set to point number 0.

:MEMORY:TABLE:POSITION:POINT

Command

:MEMory:TABLe:POSition[:POINt] <position>

<position> ::= <point_nr> | <symb_pos>
<point_nr> ::= <numeric_value>
Range 0 to ....
<symb_pos> ::= MINimum | MAXimum | TRIGger

Query

:MEMORY:TABLe:POSition[:POINt]? [<symb_pos>]

<symb_pos> ::= MINimum | MAXimum | TRIGger

Response

<point_nr>

<point_nr> ::= <NR1>
Range 0 to ...

Description

The :MEMory:TABLe:POSition:POINt command sets the position for subsequent access in the memory table to the specified point (entry) number (<point_nr>) or the point that corresponds with the specified symbolic position (<symb_pos).

The query form of this command returns the point number which corresponds with the specified symbolic position or, if omitted, the current point number.

:MEMORY:TABLE:POSITION:POINT

See also

Chapter 10: Acquisition Data Access.

Examples

Ask the point number of the trigger (i.e. the number of points before the trigger point):

→ :MEMory:TABLe:POSition:POINt? TRIGger
← 483

Ask the point number of the end of the (currently selected) table (i.e. the number of entries in the table). This returns the same value as the query :MEMORY:TABLE:POINTS?

→ :MEMory:TABLe:POSition:POINt? MAXimum
← 2048

**RST condition*

Point number is set to 0.

:MEMORY:TABLE:POSITION:STAMP

Command

:MEMory:TABLe:POSition:STAMp <position>

<position> ::= <stamp> | <symb_pos>
<stamp> ::= See Chapter 4: Introduction.
<symb_pos> ::= MINimum | MAXimum | TRIGger

Query

:MEMORY:TABLe:POSition:STAMP? [<symb_pos>]

<symb_pos> ::= MINimum | MAXimum | TRIGger

Response

<stamp>

<stamp> ::= See Chapter 4: Introduction.

Description

The :MEMory:TABLe:POSition:STAMp command sets the position for subsequent access in the memory table to the point number which corresponds with the specified timestamp (<stamp>) or with the specified symbolic position (<symb_pos>).

The query form of this command returns the timestamp which corresponds with the specified symbolic position or, if omitted, the timestamp that corresponds with the current point number.
The format and length of the returned timestamp can be specified with the :FORMat:STAMp command.

See also

Chapter 10: Acquisition Data Access.

:MEMORY:TABLE:POSITION:STAMP

Examples

Request the timestamp of the first entry in the table:

→ :MEMory:TABLE:POSition:STAMp? MINimum
← -2580

Request the timestamp of the last entry in the table:

→ :MEMory:TABLE:POSition:STAMP? MAXimum
← 7485

**RST condition*

On *RST the point number is set to 0.

:MEMORY:TABLE:POSITION:VALUE

Command

:MEMory:TABLe:POSition:VALue <val_spec>[,<chan_spec>]

<val_spec> ::= ZERO | NONZero | EQUal | DIFFer

<chan_spec> ::= <chan_nr> | <chan_list> | <label>

Default: All channels for which data has been acquired.

<chan_nr> ::= <numeric_value>

Range 1 to 96 inclusive.

<chan_list> ::= Channel list. See SCPI 1992 Volume 1 Section 8.3.2. Maximum length of 200 characters.

<label> ::= <string_data>

The channels assigned to label <label> as defined in the FORMAT menu. Maximum length of 12 characters.

Query

:MEMory:TABLe:POSition:VALue? <val_spec>[,<chan_spec>]

<val_spec> ::= See Command form.

<chan_spec> ::= See Command form.

Response

<point_nr>

<point_nr> ::= <NR1>

Range -1 to .....

:MEMORY:TABLE:POSITION:VALUE

Description

The :MEMORY:TABLE:POSITION:VALUE command searches the selected memory table (starting on the current position in forward direction) for an entry of which the sample value matches with the specified search value. The point number of the first entry that matches becomes the new position. In case of no match the position remains unchanged.

The parameter <val_spec> specifies where to search for:

ZERO : Search for a sample in which all of the specified channels has a value of 0.

NONZero : Search for a sample in which at least one of the specified channels has a value of 1.

EQUal : Is an alias for ZERO.

DIFFer : Is an alias for NONZero.

The <chan_spec> parameter defines which channels should be examined.

The query form of this command performs exactly the same with the exception that instead of updating the access position, the corresponding point number is returned.

See also

Chapter 10: Acquisition Data Access.

:MEMORY:TABLE:POSITION:VALUE

Examples

Obtain the point number of the first samples that contain a value of 1 for either channel 4, 5, 6 or 9:

```
→  :MEMory:TABLE:POSition:POINt      MINimum
→  :MEMory:TABLE:POSition:VALue?    NONZero,(@4:6,9)
←  49
```

Position on the first sample after the trigger point where the clock "CLOCK1" (for example defined in the FORMAT menu as channel 12) falls and obtain the corresponding sample number:

```
→  :MEMory:TABLe:POSition:POINt      TRIGger
→  :MEMory:TABLe:POSition:VALue      ZERO,"CLOCK1"
→  :MEMory:TABLe:POSition:NUMBer?
←  738
```

**RST condition*

At *RST the access position is set to point number 0.

Errors

-222, "Data out of range"

Invalid channel number (<chan-nr>) specified.

-223, "Too much data"

Specified channel list (<chan-list>) exceeds 200 characters.

:MEMORY:TABLE:SAMPLE:DATA

Query

:MEMory:TABLe:SAMPle[:DATA]? [<count>[,<chan_spec>]]

<count> ::= <numeric_value>
Range1 to ...
Default : 1

<chan_spec> ::= <chan_nr> | <chan_list> | <label>
Default : All channels for which data has been stored.

<chan_nr> ::= <numeric_value>
Range 1 to 96 inclusive.

<chan_list> ::= Channel list as defined in SCPI 1992 Volume 1 Section 8.3.2.
Maximum length of 200 characters.

<label> ::= <string_data>
The channels assigned to label <label> as defined in the format menu.
Maximum string length is 12 characters.

Response

[<acq_data>{,<acq_data>}]

<acq_data> ::= <stamp>,<value>,<status>

<stamp> ::= The timestamp at which the sample was taken.

<value> ::= <NR1>

<status> ::= <NR1>
Range 0 to 8 inclusive.

:MEMORY:TABLE:SAMPLE:DATA

Description

The :MEMory:TABLe:SAMPle:DATA query returns <count> acquisition data items from the selected memory table in either forward or reverse direction as specified by the :MEMory:TABLe:ACCess:FORward command.

The query starts at the current position which is incremented/decremented upon query completion with the number of items actually returned. This may be less than the number requested if the end/beginning of the table is reached before <count> items are returned.

The format and length of the time stamp can be specified with the :FORMat:STAMp command.

The format and length of the sample value can be specified with the :FORMat:DATA command.

See also

Chapter 10: Acquisition Data Access.

Examples

Read 3 items of acquired data for label BUS starting at the trigger point followed by a query for the next 2 items:

```
→   :MEMory:TABLe:POSition:POINt  TRIGger
→   :MEMory:TABLe:SAMPle:DATA?   3,"BUS"
←   2040,412,1,2050,278,1,2090,400,2
→   :MEMory:TABLe:SAMPle:DATA?   2,"BUS"
←   3010,78,2,3375,212,2
```

After this command/query sequence the current access position is TRIGger + 5.

:MEMORY:TABLE:SAMPLE:DATA

**RST condition*

Not applicable

Errors

-222, "Data out of range"

Specified number of items (<counts>) or channel number (<chan_nr>) outside legal range.

-223, "Too much data"

Length of the channel list (<chan_list>) exceeds 200 characters.

:MEMORY:TABLE:SAMPLE:DURATION

Query

:MEMory:TABLe:SAMPle:DURation? [<count>]

<count> ::= <numeric_value>
Range 1 to ...
Default: 1

Response

[<duration>{,<duration>}]

<duration> ::= <stamp>
Range 0 to ...

Description

The :MEMory:TABLe:SAMPle:DURation? query returns the duration of <count> samples in the selected memory table in either forward or reverse direction.

The query starts at the current position which is incremented/decremented upon query completion with the number of duration items actually returned. This may be less than the number requested if the end/beginning of the table is reached before <count> items are returned.

The format and length of the duration can be defined with the :FORMat:DATA command.

See also

Chapter 10: Acquisition Data access.

:MEMORY:TABLE:SAMPLE:DURATION

Examples

This example shows how to obtain the duration of the first 3 samples stored in the currently selected memory table:

→ :MEMory:TABLe:POSition:POINt 0
→ :MEMory:TABLe:SAMPle:DURation? 3
← 20,44,13

**RST condition*

Not applicable.

Errors

- 222, "Data out of range"

  Invalid specification of the number of items (<count>).

:MEMORY:TABLE:SAMPLE:STAMP

Query

:MEMory:TABLe:SAMPle:STAMp? [<count>]

<count> ::= <numeric_value>
Range 1 to ..
Default :1

Response

[<stamp>{,<stamp>}]

<stamp> ::= Timestamp at which the sample was taken.

Description

The :MEMory:TABLe:SAMPle:STAMp query returns <count> timestamps from the selected memory table in either forward or reverse direction as specified by the :MEMory:TABLe:ACCess:FORward command.

The query starts at the current position which is incremented/decremented with the number of timestamps actually returned. This may be less than the number requested if the end/beginning of the table is reached before <count> timestamps have been returned.

The format and length of the time stamp can be specified with the :FORMAT:STAMP command.

See also

Chapter 10: Aquisition Data Access

:MEMORY:TABLE:SAMPLE:STAMP

Examples

Obtain the timestamps of the first 5 samples in hexadecimal format with a minimum length of 6 hexadecimal digits:

→ :MEMory:TABLe:POSition:POINt 0
→ :FORMat:STAMp HEXadecimal,6
→ :MEMory:TABLe:SAMPle:STAMp? 5
← #H000010,#H000200,#H000205,#H000210,#H000290

Errors

-222, "Data out of range"

Invalid specification of number of items (<counts>).

:MEMORY:TABLE:SAMPLE:STATUS

Query

:MEMory:TABLe:SAMPle:STATus? [<count>]

<count> ::= <numeric_data>

Range 1 to ..

Default : 1

Response

[<status>{,<status>}]

<status> ::= <NR1>

Range 0 to 8 inclusive with the following meaning:

0 = Sample taken at sequence level 1

1 = Sample taken at sequence level 2

2 = Sample taken at sequence level 3

3 = Sample taken at sequence level 4

4 = Sample taken at sequence level 5

5 = Sample taken at sequence level 6

6 = Sample taken at sequence level 7

7 = Sample taken at sequence level 8

8 = Sample taken after trigger

Description

The :MEMory:TABLe:SAMPle:STATus query returns the status of <count> samples from the selected memory table in either forward or

:MEMORY:TABLE:SAMPLE:STATUS

reverse direction specified by the :MEMory:TABLe:ACCess:FORward command.

The query starts at the current position which is incremented/decremented with the number of status codes actually returned. This may be less than the number requested.

Examples

Chapter 10: Acquisition Data Access.

**RST condition*

Not applicable.

Errors

-222, "Data out of range"

Invalid specification of number of items (<count>).

:MEMORY:TABLE:SAMPLE:VALUE

Query

:MEMory:TABLe:SAMPle:VALue? [<count>[,<chan_spec>]]

<count> ::= <numeric_value>

Range 1 to ....

Default: 1

<chan_spec> ::= <chan_nr> | <chan_list> | <label>

Default: All channels for which data has been acquired.

<chan_nr> ::= <numeric_value>

Range 1 to 96 inclusive.

<chan_list> ::= Channel list as defined in SCPI 1992 Volume 1 Section 8.3.2.

<label> ::= <string_data>

All channels assigned to label <label> as defined in the FORMAT menu.

Response

[<value>{,<value>}]

<value> ::= <NR1>.

Description

The :MEMory:TABLe:SAMPle:VALue? query returns <count> sample values from the selected memory table in either forward or reverse direction as specified by the :MEMory:TABLe:ACCess:FORWard command. The query starts at the current position which is incremented/decremented upon query completion with the number of items actually returned. This may be less than the number request-

:MEMORY:TABLE:SAMPLE:VALUE

ed if end/beginning of the table is reached before <count> items are returned.

The format and length of the sample value can be specified with the :FORMat:DATA command.

See also

Chapter 10: Acquisition Data Access.

Examples

Read 3 sample values for channels 3, 6, 7 and 8 starting at entry number 0:

```
→   :MEMory:TABLe:POSition:POINt      0
→   :MEMory:TABLe:SAMPle:VALue?   3,(@3,6:8)
←   4,15,3
```

**RST condition*

Not applicable.

Errors

-222, "Data out of range"

Invalid specification of number of items (<count>) or invalid channel number (<chan_nr>).

-223, "Too much data"

Length of the specified channel list (<chan_list>) exceeds 200 characters.

:MEMORY:TABLE:SELECT

Command

:MEMory:TABLe:SELect <table_name>

<table_name> ::= NEW1T | NEW1S | NEW2T | NEW2S | REF1T | REF1S | REF2T | REF2S | COM1T | COM1S | COM2T | COM2S

The table name <table_name> is character data.

Query

:MEMory:TABLe:SELect?

Response

<table_name>

<table_name> ::= See command form.

Description

The :MEMory:TABLe:SELect command selects the memory table specified by <table_name> for subsequent table access commands and queries.

The query form of this command returns the name of the currently selected table.

:MEMORY:TABLE:SELECT

See also

Chapter 10: Acquisition Data Access.

SCPI 1992 Volume 2 Section 11.6.1

Examples

The following command selects STATE data of analyzer 1 from the NEW measurement for subsequent access:

→ :MEMory:TABLe:SELect NEW1S

→ :MEMory:TABLe:SELect?
← NEW1S

**RST condition*

At *RST no table is selected.

MMEMory Subsystem

:MMEMory

HEADER	PARAMETERS	RESPONSE
:**MMEM**ory:**CAT**alogue?		<disk_info>
→ :**COPY**	<from>,<to>	
→ :**DEL**ete	<file_name>	
→ :**INIT**ialize	[<label_name>]	
→ :**LOAD**:**CONF**iguration	[<file_name>]	
→ :**MEAS**urement	<file_name>{,<part>}	
→ :**SYMB**ols	<file_name>	
→ :**SYST**em		
→ :**MOVE**	<from>,<to>	
→ :**STOR**e:**MEAS**urement	<file-name>{,<part>}	
→ :**SCR**een	[<file-name>]	
→ :**TRAN**sfer	<file_name> {,<data>}	
→ :**TRAN**sfer?	<file_name>	[<data>{,<data>}]

:MMEMORY:CATALOGUE

Query

:MMEMory:CATalogue?

Response

<used>,<available>{,<file_entry>}

<used>	::=	Number of bytes in use (for files, directories, etc). Type <NR1>.
<available>	::=	Number of bytes available (free) on the diskette. Type <NR1>.
<file_entry>	::=	"<file_name>,<file_type>,<file_size>"
<file_name>	::=	Name of the file as recorded in the diskette directory.
<file_type>	::=	Because the analyzer is not able to determine the file type from directory information, this field is empty.
<file_size>	::=	Size of the file in bytes. Type <NR1>.

Description

The :MMEMory:CATalogue? query returns information about the diskette currently mounted in the analyzer's diskette drive.

See also

SCPI 1992 Volume 2 Section 12.1

Examples

→ :MMEMory:CATalogue?
← 791040,666624,"SYSTEM,,783104","USER.CFG,,7270"

:MMEMORY:CATALOGUE

Errors

-252, "Missing media"

No diskette mounted.

**RST condition*

Not applicable.

:MMEMORY:COPY

Command

:MMEMory:COPY <src_file_name>,<dst_file_name>

<src_file_name> ::= <file_name>

<dst_file_name> ::= <file_name>

Description

The :MMEMory:COPY command copies the contents of the file named by <src_file_name> to a new file with the name <dst_file_name>.
The execution of this command is suspended if a local dialogue is in progress or the diskette is currently being used for local operation.

See also

Reference Guide: Input/Output Menu-Copy

SCPI 1992 Volume 2 Section 12.3

Examples

→ :MMEMory:COPY "MEAS.001","MEAS.AUT"

**RST condition*

Not applicable.

:MMEMORY:DELETE

Command

:MMEMory:DELete <file_name>

Description

The :MEMory:DELete command removes the file specified by <file_name> from the diskette.
The execution of this command is suspended if a local dialogue is in progress or the diskette is currently being used for local operation.

See also

Reference Guide: Input/Output Menu-Delete.

SCPI 1992 Volume 2 Section 12.4

Examples

→ :MMEMory:DELete "SCREEN.001"

**RST condition*

Not applicable.

:MMEMORY:INITIALIZE

Command

:MMEMory:INITialize [<label_name>]

<label_name> ::= Diskette label name of maximum 11 characters.
default : "PM3580-3585"

Description

The :MMEMory:INITialize command initializes the diskette.

See also

SCPI 1992 Volume 2 Section 12.5

Examples

→ :MMEMory:INITitialize "backup"

**RST condition*

Not applicable.

:MMEMORY:LOAD:CONFIGURATION

Command

:MMEMory:LOAD:CONFiguration [<file_name>]

<file_name> ::= Name of the file containing the configuration settings.
default : "USER.CFG"

Description

The :MMEMory:LOAD:CONFiguration command loads the contents of the configuration file specified by <file_name> in the analyzer's memory.

See also

PM 3580/PM 3585 User Manual Chapter 10

SCPI 1992 Volume 2 Section 12.6

Examples

→ :MMEMory:LOAD:CONFiguration "A84713.CFG"

**RST condition*

Not applicable

Errors

-223,"Too much data"
<file-name> more than 12 characters.

:MMEMORY:LOAD:MEASUREMENT

Command

:MMEMory:LOAD:MEASurement <file_name>{,<part>}

<file_name> ::= Name of the measurement file.
<part> ::= NEW | REFerence | SETTings | ALL
default : ALL

Description

The :MMEMORY:LOAD:MEASUREMENT command loads the contents of the measurement file specified by <file_name> in the analyzer's memory.

The execution of this command is suspended if a local dialogue is in progress.

The <part> parameter specifies which part(s) have to be extracted from the file.

See also

PM3580/3585 Reference Guide : Input/Output Menu Load

SCPI 1992 Volume 2 Section 12.6

Examples

→ :MMEMory:LOAD:MEASurement "MEAS.001",NEW,SETtings

→ :MMEMory:LOAD:MEASurement "MEAS.AUT",REFerence

**RST condition*

Not applicable

:MMEMORY:LOAD:MEASUREMENT

Errors

-223, "Too much data"

<file_name> more than 12 characters.

:MMEMORY:LOAD:SYMBOLS

Command

:MMEMory:LOAD:SYMBols <file_name>

<file_name> ::= Name of the file containing the symbol definitions.

Description

The :MMEMory:LOAD:SYMBols command loads the symbols from the symbol file specified by <file_name> in the analyzer's memory. The execution of this command is suspended if a local dialogue is in progress.

See also

User Manual chapter 10: Symbol File.

Reference Guide : Input/Output Menu

SCPI 1992 Volume 2 Section 12.6

Examples

→ :MMEMory:LOAD:SYMBols "VIDEO.SYM"

**RST condition*

Not applicable

Errors

-223, "Too much data"

<file_name> more than 12 characters.

:MMEMORY:LOAD:SYSTEM

Command

:MMEMory:LOAD:SYSTem

Description

The :MMEMory:LOAD:SYSTem command loads the contents of the file "SYSTEM" from diskette in memory.

This command has the same effect as a system reboot.

See also

Reference Guide : System Boot.

SCPI 1992 Volume 2 Section 12.6

Examples

→ :MMEMORY:LOAD:SYSTEM

:MMEMORY:MOVE

Command

:MMEMory:MOVE <src_file_name>,<dst_file_name>

<sre_file_name> ::= <file_name>
Current file name.

<dst_file_name> ::= <file_name>
New file name.

Description

The :MMEMory:MOVE command moves (changes the name of) the file <src_file_name> to <dst_file_name>.

If <dst_file_name> already exists, it is removed before <src_file_name> is moved.

The execution of this command is suspended if a local dialogue is in progress or the diskette is currently being used for local operation.

See also

Reference Guide: Input/Output Menu

SCPI 1992 Volume 2 Section 12.8

Examples

→ :MMEMory:MOVE "FILE.OLD","FILE.NEW"

**RST condition*

Not applicable.

:MMEMORY:STORE:MEASUREMENT

Command

:MMEMory:STORe:MEASurement <file_name>{,<part>}

<file_name> ::= Name of the file to be created.
<part> ::= NEW | REFerence | SETTings | ALL
default : ALL

Description

The :MMEMory:STORe:MEASurement command saves measurement and settings data in a diskette file named <file_name>.

The optional <part> parameter(s) specify what type of data should be stored on diskette.

The execution of this command is suspended if a local dialogue is in progress or the diskette is currently being used for local operation.

See also

:MMEMory:LOAD:MEASurement command
Reference Guide: Input/Output Menu - Save

Examples

The following command saves the NEW acquisition data and the settings in the file "MEAS.001":

→ :MMEMory:STORe:MEAS "MEAS.001",NEW,SETTings

Any of the following commands will save NEW acquisition data, REFerence data and the settings in the file "MEAS.002":

→ :MMEMory:STORe:MEAS "MEAS.002"

→ :MMEMory:STORe:MEAS "MEAS.002",ALL

→ :MMEMory:STORe:MEAS "MEAS.002",NEW,REF,SETT

:MMEMORY:STORE:MEASUREMENT

**RST condition*

Not applicable

:MMEMORY:STORE:SCREEN

Command

:MMEMory:STORe:SCReen [<file_name>]

<file_name> ::= Name of the screen hardcopy file.
default : "SCREEN.001"

Description

The :MMEMory:STORe:SCReen command writes the current screen image to the file <file_name>.

See also

User Manual Chapter 10: Hardcopy File.

Examples

→ :MMEMory:STORe:SCReen "SCREEN.002"

**RST condition*

Not applicable.

:MMEMORY:TRANSFER

Command

:MMEMory:TRANsfer <file_name> {,<data>}

Query

:MMEMory:TRANsfer? <file_name>

Response

[<data>{,<data>}]

Description

The :MMEMory:TRANsfer command transfers data from the controller to the analyzer diskette file specified by <file_name>.

The :MMEMory:TRANsfer query transfers the contents of the file specified by <file_name> from the analyzer diskette to the controller, as a sequence of definite length arbitrary data blocks.

Examples

→ :MMEMory:TRANsfer "MEAS.AUT", <data>,....,<data>

→ :MMEMory:TRANsfer? "MEAS.001"

← <block>,<block>,...

**RST condition*

Not applicable.

:MMEMORY:TRANSFER

SENSe Subsystem

:SENSe

HEADER	PARAMETERS	RESPONSE
:**SENS**e[:**CHAN**nel]:**ACT**ivity?	<channels>	<activity>{,<activity>}

:SENSE:CHANNEL:ACTIVITY

Query

:SENSe[:CHANnel]:ACTivity? <channels>

<channels> ::= <chan_nr> | <chan_list>
<chan_nr> ::= <numeric_value>
Range 1 to 96 inclusive.
<chan_list> ::= See SCPI 1992 Volume 1 Section 8.3.2

Response

<activity>{,<activity>}

<activity> ::= 0 | 1 | 2
0 = Low
1 = High
2 = Changing

Description

The :SENSe:CHANnel:ACTivity? query returns an indication about the activity on each specified channel.

Examples

→ :SENSe:CHANnel:ACTivity? 12
← 2

→ :SENSe:CHANnel:ACTivity? (@4:7,9,11:13)
← 1,0,0,2,2,0,2,2

:SENSE:CHANNEL:ACTIVITY

**RST condition*

Not applicable.

Errors

-222, "Data out of range"

Specified channel outside range of 1 to 96.

-223, "Too much data"

<chan_list> exceeds 203 characters

-241, "Hardware missing"

Specified channel not in hardware configuration

STATus Subsystem

:STATus

HEADER	PARAMETERS	RESPONSE
***CLS**		
***ESE**	<enable_mask>	
***ESE**?		<enable_mask>
***ESR**?		<event_status>
***SRE**	<enable_mask>	
***SRE**?		<enable_mask>
***STB**?		<status_byte>
:**STAT**us:**OPER**ation:**COND**ition?		<status>
→ [:**EVEN**t]?		<events>
→ :**ENAB**le	<mask>	
→ :**ENAB**le?		<mask>
→ :**NTR**ansition	<filter>	
→ :**NTR**ansition?		<filter>
→ :**PTR**ansition	<filter>	
→ :**PTR**ansition?		<filter>
→ :**PRES**et		
→ :**QUES**tionable:**COND**ition?		<status>
→ :**ENAB**le	<mask>	
→ :**ENAB**le?		<mask>
→ [:**EVEN**t]?		<events>
→ :**NTR**ansition	<filter>	
→ :**NTR**ansition?		<filter>
→ :**PTR**ansition	<filter>	
→ :**PTR**ansition?		<filter>
→ :**QUE**ue[:**NEXT**]?		<code>,<text>

*CLS

Command

*CLS

Description

The *CLS (Clear Status) command performs the following:

- Clears all event registers which are summarized in the Status Byte register (STB). These event registers are:
  - The Standard Event Status register (ESR)
  - The Operation Event Status register.
  - The Questionable Event Status register
- Empties all Queues which are summarized in the Status Byte register (STB) with the exception of the Output Queue. The following queues are affected:

  -The Error/Event Queue.

  -The Key Queue.
- Forces the analyzer into the Operation Complete Command Idle State and the Operation Command Query Idle State.

See also

IEEE-488.2 1987 Sections 10.3 and 11.1.2

Examples

→ *CLS

*ESE

Command

***ESE <value>**

<value> ::= <numeric_data>

Query

***ESE?**

Response

<value>

<value> ::= <NR1>

Description

The *ESE (standard Event Status Enable) command sets the standard Event Status Enable register to the value specified by <value> in order to enable the events recorded in the standard Event Status register (ESR) to be summarized in the Status Byte register (STB).

The *ESE query returns the contents of the standard Event Status Enable register, indicating which events in the standard Event Status register (ESR) should be summarized in the Status Byte register (STB).

The output format of the *ESE query can be selected with the FORMat:SREGister command.

See also

Chapter 5: Status Reporting System.

IEEE-488.2 1987 Section 10.10 and 10.11.

*ESE

Examples

Each of the following commands enables the OPC (Operation Complete) and the URQ (User Request) bit to be summarized in the Status Byte register (STB).

This is done by setting bit 0 and bit 6 in the ESE register.

→ *ESE #B01000001

← *ESE 65

The following query shows that bit 2 (QYE: Query Error) and 5 (CME: Command Error) are set. This means that when a query error or a command error event occurs, the summary bit 5 (ESB) in the Status Byte register (STB) will be set.

→ *ESE?
← #B00100100

*ESR

Query

*ESR?

Response

<value>

<value> ::= <NR1>

Description

The *ESR query returns the contents of the standard Event Status Register. Reading the ESR register clears it.

The command :FORMat:SREGister can be used to select output format of this query

See also

Chapter 5: Status Reporting System.

IEEE-488.2 1987 Section 10.12.

Examples

→ *ESR?

← #B00011100

Bit 4 (EXE), bit 3 (DDE) and bit 2 (QYE) are set. This means that since the last time this register was read, an Execution Error, a Device Dependent Error and a Query Error have occurred.

Note that the next query will return the value 0, provided that no new events have occurred.

*SRE

Command

***SRE <value>**

<value> ::= <numeric_data>

Query

***SRE?**

Response

<value>

<value> ::= <NR1>

Description

The *SRE (Service Request Enable) command sets the Service Request Enable register to the value specified by <value>.

The *SRE query returns the contents of the SRE register. Note that bit 6 is always 0.
The response format (ASCii, BINary, etc) can be defined with the FORMat:SREGister command.

The SRE register determines:

- Which corresponding bits in the Status Byte register (STB) will cause a service request from the analyzer.
- Which corresponding bits in the Status Byte register (STB) will be summarized in the MSS-bit of this register.

See also

Chapter 5: Status Reporting System.

IEEE-488.2 1987 Section 10.34 and 10.35.

*STB

Query

***STB?**

Response

<value>

<value> ::= <NR1>

Description

The *STB (Status Byte) query returns the Status Byte and Master Summary Status bit.
The command :FORMat:SREGister can be used to select the format of the response.

See also

Chapter 5: Status Reporting System.

IEEE-488.2 1987 Section 10.36.

:STATUS:OPERATION:CONDITION

Query

:STATus:OPERation:CONDition?

Response

<condition>

<condition> ::= <NR1>

Description

The :STATus:OPERation:CONDition query returns the contents of the Operation Condition register. The contents of the register remains unchanged.

The command :FORMat:SREGister can be used to select the format of the response.

See also

Chapter 5: Status Reporting System.

SCPI 1992 Volume 2 Section 18.2

Examples

→ :STATus:OPERation:CONDition?
← #B0000110100010000

:STATUS:OPERATION:ENABLE

Command

:STATus:OPERation:ENABle <mask>

<mask> ::= <numeric_data>

Query

:STATus:OPERation:ENABle?

Response

<mask>

<mask> ::= <NR1>

Description

The :STATus:OPERation:ENABle command sets the enable mask in the Operation Event Enable register. In order to enable/disable the events, recorded in the Operation Event register to be summarized in bit 7 (OPER) of the Status Byte register (STB).
If a bit is 1 in the enable register and its corresponding bit in the event register transitions to 1, bit 7 (OPER) in the Status Byte register will be set to 1.

See also

Chapter 5: Status Reporting System.

SCPI 1992 Volume 2 Section 18.3

:STATUS:OPERATION:EVENT

Query

:STATus:OPERation[:EVENt]?

Response

<events>

<events> ::= <NR1>

Description

The :STATus:OPERation:EVENt query returns the contents of the Operation Event register. Reading the event register will clear its contents.

The command :FORMat:SREGister can be used to select the response format of this query.

See also

Chapter 5: Status Reporting System.

SCPI 1992 Volume 2 Section 18.1.

:STATUS:OPERATION:NTRANSITION

Command

:STATus:OPERation:NTRansition <filter>

<filter> ::= <numeric_data>

Query

:STATus:OPERation:NTRansition?

Response

<filter>

<filter> ::= <NR1>

Description

The :STATus:OPERATION:NTRANSITION command sets the Negative Operation Status Transition filter.

For each bit set in the negative transition filter, a 1 to 0 transition of the corresponding bit of the Operation Condition register causes a 1 to be set in the corresponding bit of the Operation Event register.

The query form returns the contents of this register. The output format can be selected with the :FORMat:SREGister command.

See also

Chapter 5: Status Reporting System.

SCPI 1992 Volume 2 Section 18.5.

:STATUS:OPERATION:PTRANSITION

Command

:STATus:OPERation:PTRansition <filter>

<filter> ::= <numeric_data>

Query

:STATus:OPERation:PTRansition?

Response

<filter>

<filter> ::= <NR1>

Description

The :STATus:OPERation:PTRansition command sets the Positive Operation Transition filter.

For each bit set in the positive transition filter, a 0 to 1 transition of the corresponding bit of the Operation Condition register causes a 1 to be set in the corresponding bit of the Operation Event register.

The query form returns the contents of this register. The output format can be selected with the :FORMat:SREGister command.

See also

Chapter 5: Status Reporting System.

SCPI 1992 Volume 2 Section 18.4.

:STATUS:PRESET

Command

:STATus:PRESet

Description

The :STATus:PRESet command presets the Operation and Questionable status data structures. It affects only the enable register and transition filters for these structures as follows:

- The Operation/Questionable event enable register is set to 0.
- The Operation/Questionable positive transition filter is set to 7FFF hexadecimal.
- The Operation/Questionable negative transition filter is set to 0.

See also

Chapter 5: Status Reporting System.

SCPI 1992 Volume 2 Section 18.7.

:STATUS:QUESTIONABLE:CONDITION

Query

:STATus:QUEStionable:CONDition?

Response

<condition>

<condition> ::= <NR1>

Description

The :STATus:QUEStionable:CONDition query returns the contents of the Questionable Condition Register. Reading the condition register does not change its contents.

The :FORMat:SREGister command can be used to select the response format of this query.

See also

Chapter 5: Status Reporting System.

SCPI 1992 Volume 2 Section 18.2

Examples

→ :STATus:QUEStionable:CONDition?
← #B0000000100000000

:STATUS:QUESTIONABLE:ENABLE

Command

:STATus:QUEStionable:ENABle <mask>

<mask> ::= <numeric_data>.

Query

:STATus:QUEStionable:ENABle?

Response

<mask>

<mask> ::= <NR1>

Description

The :STATus:QUEStionable:ENABle command sets the enable mask in the Questionable Event Enable register. This register is used to enable/disable the events, recorded in the Questionable Event register, to be summarized in bit 3 (QUES) of the Status Byte register (STB).
If a bit is 1 in the enable register and its corresponding bit in the event register changes to 1, bit 3 (QUES) is the Status Byte register will be set to 1.

See also

Chapter 5: Status Reporting System.

SCPI 1992 Volume 2 Section 18.3

:STATUS:QUESTIONABLE:EVENT

Query

:STATus:QUEStionable[:EVENt]?

Response

<events>

<events> ::= <NR1>

Description

The :STATus:QUEStionable:EVENt query returns the contents of the Questionable Event register. Reading the event register will clear its contents.

The :FORMat:SREGister command can be used to select the response format of this query..

See also

Chapter 5: Status Reporting System.

SCPI 1992 Volume 2 Section 18.1.

:STATUS:QUESTIONABLE:NTRANSITION

Command

:STATus:QUEStionable:NTRansition <filter>

<filter> ::= <numeric_data>

Query

:STATus:QUEStionable:NTRansition?

Response

<filter>

<filter> ::= <NR1>

Description

The :STATus:QUEStionable:NTRansition command sets the Negative Questionable Transition filter.

For each bit set in the negative transition filter a 1 to 0 transition of the corresponding bit in the Questionable Condition register causes a 1 to be set in the corresponding bit of the Questionable Event Register.

The query form returns the contents of this register. The output format can be selected with the :FORMat:SREGister command.

See also

Chapter 5: Status Reporting System.

SCPI 1992 Volume 2 Section 18.5.

:STATUS:QUESTIONABLE:PTRANSITION

Command

:STATus:QUEStionable:PTRansition <filter>

<filter> ::= <numeric_data>

Query

:STATus:QUEStionable:PTRansition?

Response

<filter>

<filter> ::= <NR1>

Description

The :STATus:QUEStionable:PTRansition command sets the Positive Questionable Transition filter.

For each bit set in the positive transition filter a 0 to 1 transition of the corresponding bit of the Questionable Condition Register causes a 1 to be set in the corresponding bit of the Questionable Event Register.

The query form returns the contents of this register. The output format can be selected with the :FORMat:SREGister command.

See also

Chapter 5: Status Reporting System.

SCPI 1992 Volume 2 Section 18.4.

:STATUS:QUEUE:NEXT

Query

:STATus:QUEue[:NEXT]?

Response

<error_number>, <error_text>

<error_number> ::= <NR1>

<error_text> ::= <string_data>

Description

The :STATus:QUEue:NEXT query returns the next entry from the analyzer's error/event queue.

This query is an alias to the :SYSTEM:ERROR? query.

See also

Chapter 6: Error and Event Reporting.

SCPI 1992 Volume 2 Sections 18.8.1 and 19.7.

Examples

→ :STATus:QUEUe:NEXT?
← -103,"Invalid separator"

→ :STATus:QUEue:NEXT?
← 0,"No error"

**RST condition*

Not applicable.

SYNChronization Subsystem

:SYNChronisation

HEADER	PARAMETERS	RESPONSE
*OPC?	None	<NR1>
*WAI	None	None

*OPC

Command

*OPC

*OPC?

Response

1

Description

The *OPC (Operation Complete) command causes the analyzer to generate the operation complete message in the Standard Event Status register when all pending selected device operations have been finished.

The *OPC query places an ASCII character 1 into the analyzer's Output Queue when all pending selected device operations have been finished.

See also

Chapter 8: Command Synchronization.

IEEE-488.2 1987 Sections 10.18, 10.19, 12.5.2 and 12.5.3

*WAI

Command

*WAI

Description

The *WAI (Wait-to-Continue) command prevents the analyzer from executing any further commands or queries until the NO-Operation-Pending flag is TRUE (i.e. all previous commands and queries have been completed).

See also

*OPC Command and Query

Chapter 8: Command Synchronization.

IEEE-488.2 1987 Sections 10.39 and 12.5.1

SYSTem Subsystem

:SYSTem

HEADER	PARAMETERS	RESPONSE
*IDN?		<identification>
*OPT?		{<option>}
*RST		
:SYSTem:BOOT		
:COMMunicate:GPIB:ADDRess	<address>	
:ADDRess?		<address>
:DATe	<date>	
:DATe?		<date>
:ERRor?		<error>
:KEY	<code>{,<code>}	
:KEY?		<code>
:KLOCk	<state>	
:KLOCk?		<state>
:PRESet		
:TIME	<time>	
:TIME?		<time>
:VERSion?		<version>

*IDN

Query

*IDN?

Response

PHILIPS,PM358<x>/<y><z>,0,V1.0

<x> ::= Frequency

0 = 100Mhz
5 = 200Mhz

<y> ::= Number of channels

3 = 32 channels
6 = 64 channels
9 = 96 channels

<z> ::= IEEE option

0 = No IEEE installed.
1 = IEEE installed.

Description

The *IDN (Identification) query returns the identification of the analyzer as an <ARBITRARY ASCII RESPONSE DATA> element.

See also

IEEE-488.2 1987: Section 10.14.1

Examples

→ *IDN?

← PHILIPS,PM3585/91,0,V1.0

*OPT

Query

***OPT?**

Response

[<option>{,<option>}]

<option> ::= IEEE | RS232

Description

The *OPT (Option Identification) query returns a list of options installed in the analyzer.

See also

IEEE-488.2 1987: Section 10.20

Examples

→ *OPT?

← IEEE,RS232

*RST

Command

*RST

Description

The *RST command resets the analyzer.

The execution of this command is suspended if a local dialogue is in progress.

See also

IEEE-488.2 1987: Section 10.32

:SYSTEM:BOOT

Command

:**SYSTem:BOOT**

Description

The :SYSTem:BOOT command reboots the analyzer.
This command has the same effect as switching the power button off and on.

See also

Command :MMEMory:LOAD:SYSTem

**RST condition*

Not applicable.

:SYSTEM:COMMUNICATE:GBIB:ADDRESS

Command

:SYStem:COMMUnicate:GPIB:ADDRess <address>

<address> ::= <numeric_data>

Range 0 to 30 inclusive.

Query

:SYStem:COMMUnicate:GPIB:ADDRess?

Response

<address>

<address> ::= <NR1>

Description

The :SYSTem:COMMUnicate:GPIB:ADDRess command sets the GPIB port address of the analyzer to the value specified by <address>.

The :SYSTem:COMMUnicate:GPIB:ADDRess? query returns the current GPIB port address.

See also

SCPI 1992 Volume 2 Section 19.3.2.1.

PM 3580/PM 3585 Reference Guide : I/O Menu

Examples

→ :SYSTEM:COMMUNICATE:GPIB:ADDRESS 3

→ :SYSTEM:COMMUNICATE:GPIB:ADDRESS?

← 3

:SYSTEM:COMMUNICATE:GBIB:ADDRESS

**RST condition*

The GPIB address is not affected by a *RST command.

Errors

-222, "Data out of range"

Specified GPIB address (<address>) not within valid range.

:SYSTEM:DATE

Command

:SYSTem:DATE <year>,<month>,<day>

<year> ::= <numeric_value>
Range 1990 to 2040 inclusive.

<month> ::= <numeric_value>
Range 1 to 12 inclusive.

<day> ::= <numeric_value>
Range 1 to the number of days in the specified month.

Query

:SYSTem:DATE? [<minmax>,<minmax>,<minmax>]

<minmax> ::= MINimum | MAXimum

Response

<year>,<month>,<day>

<year> ::= <NR1>
Range from 1990 to 2040 inclusive.

<month> ::= <NR1>
Range from 1 to 12 inclusive.

<day> ::= <NR1>
Range from 1 to 31 inclusive.

:SYSTEM:DATE

Description

The :SYSTem: DATE command sets the analyzer's date to <year>, <month> and <day>.

If no parameters have been specified, the :SYSTem: DATE? query returns the current date. Otherwise the minimum/maximum values of <year>, <month> and <day> are returned.

See also

SCPI 1992: Volume 2 Section 19.6

Examples

```
→   :SYSTem:DATE      1992,10,22
→   :SYSTem:DATE?
←   1992,10,22
→   :SYSTem:DATE?     MAXimum,MINimum,MAXimum
←   2040,1,31
```

**RST condition*

The date setting is not affected by a *RST command.

Errors

-222,"Data out of range"

Specified year, month or day, not within legal range.

:SYSTEM:ERROR

Query

:SYSTem:ERRor?

Response

<error_number>,<error_text>

<error_number> ::= <NR1>
<error_text> ::= <string_data>

Description

The :SYSTem:ERRor? query returns the next entry from the analyzer's error/event queue.

See also

Chapter 6: Error and Event Reporting.

SCPI 1992 Volume 2 Section 19.7

Examples

→ :SYSTem:ERRor?
← -224,"Illegal parameter value"

→ :SYSTem:ERRor?
← 0,"No error"

**RST condition*

Not applicable.

:SYSTEM:KEY

Command

:SYSTem:KEY <code>{,<code>}

<code> ::= <numeric_data>

Range 0 to 255 inclusive.

Query

:SYSTem:KEY?

Response

<code>

<code> ::= <NR1>

Range 0 to 255 inclusive.

Description

The :SYSTem:KEY command simulates the pressing of those local panel keys which correspond with the specified codes. The codes are the same as the hardware code for each key.

The analyzer maintains a FIFO queue in which the key codes are stored that correspond with the keys pressed at the instruments panel. Note that the KEY command does not fill entries in this queue. The empty/non-empty status of the queue is recorded in bit 0 of the Status Byte register. The :SYSTem:KEY? query returns the next item from this queue. If the queue is empty -1 is returned.

**RST condition*

The queue of panel codes is cleared by *RST; i.e. the queue is empty.

:SYSTEM:KEY

See also

Chapter 7: Panel Event Handling.

SCPI 1992 Volume 2 Section 19.9

Examples

The following command causes the DISPLAY menu to be displayed on the analyzer screen (6 = DISPLAY) and puts the cursor (active field) on the "Analyzer" field (44 = HOME):

→ :SYSTem:KEY 6,44

Errors

-222,"Data out of range"

Specified key code not within valid range.

:SYSTEM:KLOCK

Command

:SYSTem:KLOCk <state>

<state> ::= <Boolean>

Query

:SYSTem:KLOCk?

Response

<state>

<state> ::= 0 | 1

Description

The :SYSTem:KLOCk command unlocks/locks the keyboard of the analyzer. When the keyboard is locked no input can be given via the analyzer panel. The execution of this command is suspended if a local dialogue is in progress.

The :SYSTem:KLOCk? query returns the current state of the keyboard; i.e. UNLOCKED (0) or LOCKED (1).

See also

SCPI 1992 Volume 2 Section 19.10

**RST condition*

At *RST, this value is set to OFF (Keyboard not locked).

:SYSTEM:PRESET

Command

:SYSTem:PRESet

Description

The :SYSTem:PRESet command sets the analyzer to the same state as when the instrument is reset locally (CONFIG Menu). The execution of this command is suspended if a local dialogue is in progress.

See also

SCPI 1992 Volume 2 Section 19.13

**RST condition*

Not applicable.

:SYSTEM:TIME

Command

:SYSTem:TIME <hour>,<minute>,<second>

<hour> ::= <numeric_data>
Range 0 to 23 inclusive.

<minute> ::= <numeric_data>
Range 0 to 59 inclusive.

<second> ::= <numeric_data>
Range 0 to 59 inclusive.

Query

:SYSTem:TIME? [<minmax>,<minmax>,<minmax>]

<minmax> ::= MINimum | MAXimum

Response

<hour>,<minute>,<second>

<hour> ::= <NR1>
Range 0 to 23 inclusive.

<minute> ::= <NR1>
Range 0 to 59 inclusive.

<second> ::= <NR1>
Range 0 to 59 inclusive.

:SYSTEM:TIME

Description

The :SYSTem:TIME command sets the analyzer's clock to the time specified by <hour>, <minute> and <second>.

The :SYSTem:TIME? query returns the current time if no parameters have been specified. Otherwise it returns the minimum/maximum values of <hour>, <minute> and <second>.

See also

SCPI 1992 Volume 1 Section 7.2.1.2

SCPI 1992 Volume 2 Section 19.16

Examples

→ :SYSTem:TIME 13,39,54

→ :SYSTem:TIME?
← 13,39,54

→ :SYSTem:TIME? MAXimum,MINimum,MAXimum
← 23,0,59

**RST condition*

The time setting is not affected by *RST.

Errors

-222, "Data out of range"
Specified hour, minute or second not within valid range.

:SYSTEM:VERSION

Query

:SYSTem:VERSion?

Response

<version>

<version> ::= 1992.0

Description

The :SYSTem:VERSion? query returns an <NR2> formatted numeric value, corresponding to the SCPI version number on which the analyzer commands and queries are based.

See also

SCPI 1992 Volume 2 Section 19.18

Examples

→ :SYSTem:VERSION?
← 1992.0

**RST condition*

Not applicable.

TEST Subsystem

:TEST

HEADER	PARAMETERS	RESPONSE
*TST	None	<result>

*TST

Query

***TST?**

Response

<result>

<result> ::= ASCII character 0

Description

The *TST query has been defined by the IEEE-488.2 standard to cause the instrument to perform an internal self-test and report whether or not it detected any errors.
The PM 3580/PM 3585 main software has (currently) no self-tests. Because *TST is a mandatory query it is implemented and will always return 0.

See also

IEEE-488.2 Section 10.38

Examples

→ *TST?
← 0

**RST condition*

Not applicable.

TRIGger Subsystem

:TRIGger

HEADER	PARAMETERS	RESPONSE
:**ABOR**t	None	Not applicable
:**INIT**iate	None	Not applicable
:**STOP**	None	Not applicable

:ABORT

Command

:**ABORt**

Description

The :ABORt command terminates the current acquisition and returns the analyzer to the acquisition IDLE state. Data already acquired is not stored in the NEW acquisition memory; i.e. the contents of NEW remains unchanged.

If the acquisition is not running, this command has no effect.

See also

Chapter 9: Acquisition Control.

SCPI 1992 Volume 2 Section 22.5

Examples

→ :ABORt

**RST condition*

Not applicable.

Errors

None

:INITIATE

Command

:**INITiate**

Description

The :INITiate command starts a new acquisition.

This command is implemented as an overlapped command (See IEEE-488.2 1987: Section 12); i.e. after the measurement is started further commands are accepted.

As soon as the acquisition stops (Trigger condition detected) the analyzer returns to the acquisition IDLE state and the acquired data is stored in the NEW measurement.

See also

Chapter 9: Acquisition Control.

SCPI 1992 Volume 2 Section 22.4

Examples

→ :INITiate

Errors

-213, "Init ignored"
When the :INITiate command was given, the analyzer wasn't in the IDLE state.

-300, "Device-specific error"
Neither analyzer1 nor analyzer2 is logically switched on (See CONFIG Menu).

**RST condition*

Not applicable

:STOP

Command

:**STOP**

Description

The :STOP command stops the acquisition (manual trigger) and the data already acquired is stored in NEW. If the acquisition is not running (in IDLE state), this command has no effect.

See also

Chapter 9: Acquisition Control.

Examples

→ :STOP

Errors

None

**RST condition*

Not applicable

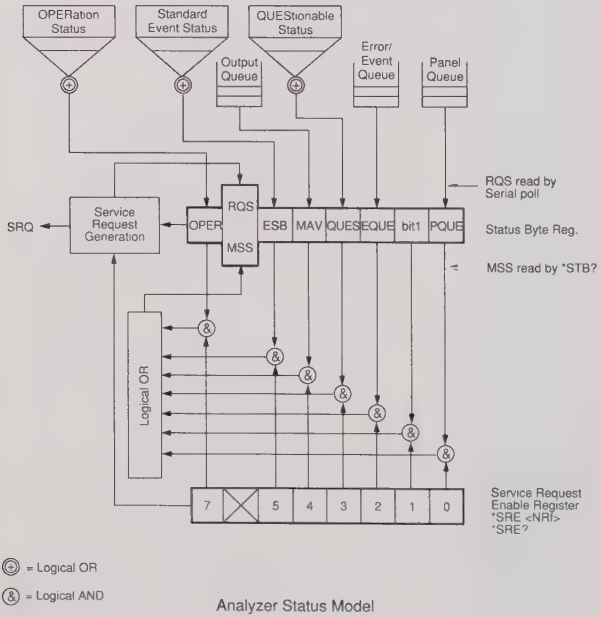
Chapter 5
Status Reporting System

Analyzer Status 5-2
Standard Status 5-6
    ESR register 5-7
    ESE register 5-8
    STB register 5-8
    SRE register 5-10
Operation/Questionable Status 5-11
    Condition Registers 5-12
    Condition Filters 5-13
    Event Register 5-14
    Event Enable Register 5-14

The status structure of the analyzer is used to reflect the current analyzer condition, and to report various events that have happened. The various status models are ultimately reported in the instruments Status Byte Register, giving a complete survey of the status of the instrument. The status structure consists of Register Model and Queue Model structures.

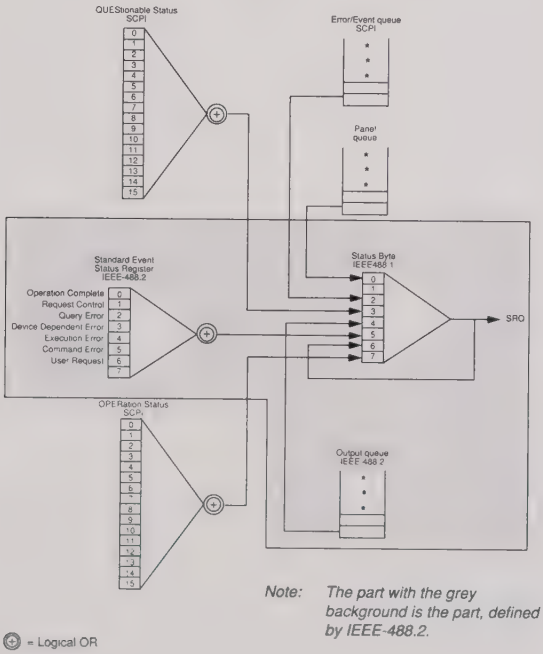
Analyzer Status

The diagram below shows the interrelations between the various components of the Analyzer Status Model.



Analyzer Status Model

All Status Models, registers and queues, are reported in one of the summary bits of the Status Byte Register. The Service Request Enable Register selects which Status Byte bits will cause a Service Request (SRQ) on the GPIB. From the Status Byte Register, which is returned as a response to a serial poll from a controller, it is always possible to trace back to the event that caused the corresponding summary bit to be set.



Note: The part with the grey background is the part, defined by IEEE-488.2.

The Status reporting system via the Status Byte Register is built on the following underlying structures:

- Register Models:
  - The OPERational status model
  - The QUEStionable status model
  - The Standard status model
- Queue Models:
  - The Output queue model
  - The Error/Event queue model
  - The Panel queue model

OPERation status This register contains the status conditions that are part of the instruments normal operation.
E.g. bit 4: Measuring.

QUEStionable status This register contains indications about the quality of instrument functions.
E.g. bit 8: Calibration (failed).

Standard status This register contains the events that are standardized by IEEE-488.2. Common commands are provided to access the register.
E.g. The standard event OPeration Complete (OPC), indicating whether an operation is completed or still busy.

Output Queue The non-empty status of the Output Queue is reported in the MAV-bit (Message AVailable) of the Status Byte Register.

Error/Event Queue The non-empty status of the Error/Event Queue is reported in bit 2 of the Status Byte Register.

Panel Queue The non-empty status of the Panel Queue is reported in bit 0 of the Status Byte Register

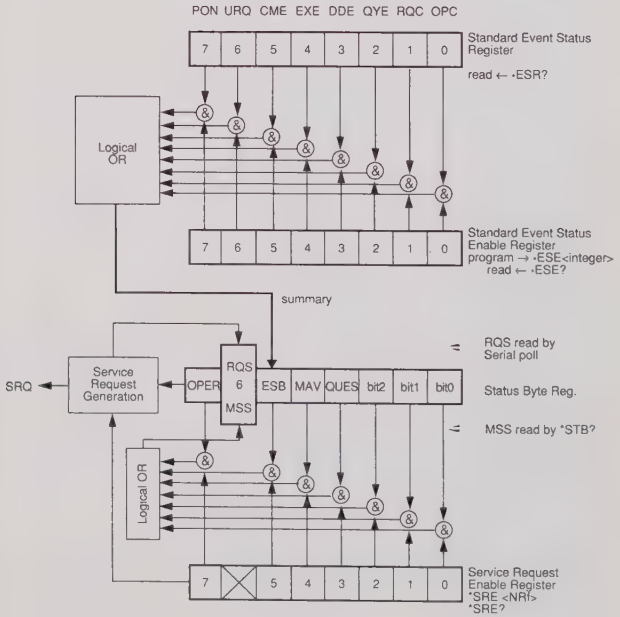
- For the Standard Status structure, there is an Event register which reflects the events itself, and an Event Enable register. The Event Enable register selects which events are summarized into the Status Byte Register.
- The Status Byte Register can be read by either Serial Poll or by the *STB? query. The Status Byte Register is cleared only by clearing its overlaying structures.
- The Service Request Enable Register allows you to select particular bits from the Status Byte to generate a Service Request (SRQ) on the GPIB.

Standard Status

The standard event status model, as defined by IEEE-488.2, consists of the following four registers:

ESR: the Standard Event Status register
ESE: the Standard Event Status Enable register
STB: the Status Byte register
SRE: the Service Request Enable register

The relation between these four registers is shown in the following diagram:



Standard Status Model

ESR register

The ESR (Event Status) register consists of 8 bits for the following standard errors/events:

bit 0 : OPC (**OP**eration **C**omplete)

OPC indicates that the analyzer has completed all previously started actions.

bit 1 : RQC (**R**e**Q**uest **C**ontrol)

RQC indicates that the instrument wants to become the active controller-in-charge.

bit 2 : QYE (**Q**uer**Y** **E**rror)

bit 3 : DDE (**D**evice **D**ependent **E**rror)

bit 4 : EXE (**EX**ecution **E**rror)

bit 5 : CME (**C**o**M**mand **E**rror)

bit 6 : URQ (**U**ser **R**e**Q**uest)

URQ indicates that the user of the device requests service. For the analyzer this means that the user has pressed a button or turned the dialknob on the analyzer's front panel.

bit 7 : PON (**P**ower **ON**)

PON indicates that the device power supply has been turned off and on since the last time the register was read or cleared.

The contents of the ESR register can be read with the *ESR? query. Reading the ESR register clears it. The contents of this register cannot be programmed; i.e. there is no *ESR command.

ESE register

The ESE (Event Status Enable) register can be used to define which standard events in the ESR register must be summarized in bit 5 (ESB) bit of the STB register.

The contents of the ESE register can be set with the *ESE command and read with the *ESE? query.

STB register

The STB (Status Byte) register contains a summary status of all overlaying status registers and queues:

bit 0 : DDS0 (**D**evice **D**ependent **S**tatus **0**)

This bit is reserved for device dependent status data that is not reported through the standard status structures. The analyzer uses this bit to indicate that at least one keycode is present in the Key Code Queue.

bit 1 : DDS1 (**D**evice **D**ependent **S**tatus **1**)

This bit is reserved for device dependent status data that is not reported through the standard status structures. This bit is not used by the analyzer (always 0).

bit 2 : EEQ (**E**rror/**E**vent **Q**ueue)

Indicates that at least one error/event message is available in the Error Event Queue.

bit 3 : QUES (**QUE**stionable **S**tatus)

Summary bit of the questionable status register structure.

bit 4 : MAV (**M**essage **AV**ailable)

Indicates whether the Output Queue contains at least one message (1) or is empty (0).

bit 5 : ESB (**E**vent **S**ummary **B**it)

Summary bit of the standard Event Status Register (ESR).

bit 6 : RQS (**Re**Quest **S**ervice)

Indicates that the device requests for service; i.e. SRQ=1 in the GPIB interface. It differs from the MSS bit in that the RQS bit is cleared after a serial poll. It is only set to 1 again when a new event occurs that requires service (as indicated in the Service Request Enable register).

bit 6 : MSS (**M**aster **S**ummary **S**tatus)

Indictes that there is an event that caused the device to request for service. The MSS bit is cleared when the event(s) in the overlaying Status Structure, that caused the Service Request, are cleared.

bit 7 : OPER (**OPER**ation Status)

Summary bit of the Operation Status register structure.

The contents of the STB register can be read in two ways:

1. With the *STB? query. This query returns the MSS bit in bit 6 of the STB register. The *STB? query does not affect the contents of the STB register.
2. With a Serial Poll. In this case the RQS bit is returned in bit 6 of the STB register. A serial poll does not affect the contents of the STB register, with the exception of the RQS bit, which is cleared after a serial poll.

Bits in the STB register (except RQS) are only cleared when the overlaying status structure that causes the summary bit being set, is cleared.

The contents of the STB register cannot be programmed; i.e. there is no *STB command.

SRE register

The SRE (Service Request Enable) register determines which corresponding bits in the Status Byte register (STB):

- will cause a service request from the analyzer
- will be summarized in the MSS bit in the STB register.

The contents of the SRE register is set with the *SRE command and read with the *SRE? query. Bit 6 in the response is always 0.

Operation/ Questionable Status

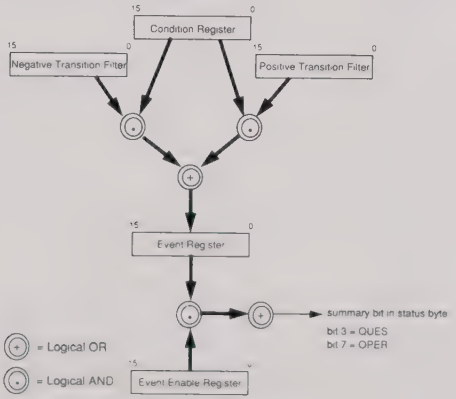
In addition to the Standard status model, as described in the previous section, two other status models are provided:

- The Operational Status model
- The Questionable Status model

Each of these models, as defined by the SCPI standard, consists of the following five registers:

- Condition register
- Negative Transition Filter register
- Positive Transition Filter register
- Event register
- Event Enable register

The following diagram shows the structure of both, the Operation and Questionable Status reporting model:



Operation and Questionable Status Model

Condition Registers

Instrument Condition registers typically reflect the instrument state. Bits in this type of register are only affected by changes of the instrument state. These registers cannot be programmed or cleared by remote messages; they can only be read by a query.

The Operation Condition register consists of 16 bits with the following meaning:

bit 0 - 3 : Not used

bit 4 : Measuring Summary

The value of this bit is the logical OR of bit 10 and bit 11; i.e. this bit is 1, if at least one logical instrument is currently measuring.
See also: Chapter 9 Acquisition Control.

bit 5 - 7 : Not used.

bit 8 : Local dialog in progress

This bit is 1 if the analyzer is currently executing a dialog via the local screen and front panel; e.g. field edit, list selection, etc..

bit 9 : Not used.

bit 10 : Measuring Analyzer 1.

This bit is 1 if logical instrument 1 is currently measuring.

bit 11 : Measuring Analyzer 2.

This bit is 1 if logical instrument 2 is currently measuring.

bit 12-15 : Not used.

The contents of the Operation Condition register can be read with the STATus:OPERation:CONDition? query. Reading this register does

not affect its contents. The register cannot be programmed; i.e. there is no STATUS:OPERATION:CONDITION command.

The Questionable Condition register consists of 16 bits with the following purpose:

bit 0 - 7 : Not used

bit 8 : Calibration

This bit indicates that the calibration function, executed at system power on, failed.

bit 9 -15 : Not used

The contents of the Questionable Condition register can be read with the STATus:QUEStionable:CONDition? query. Reading this register does not affect its contents. The register cannot be programmed; i.e. there is no STATus:QUEStionable:CONDition command.

Condition Filters

Instrument conditions can be filtered before they are reported as events in one or both of the following ways:

- Reported when getting valid (from 0 -> 1), i.e. when a positive transition of the condition occurs. The Positive Transition Filter selects which positive transitions are reported in the Event Register.
- Reported when getting non-valid (from 1 -> 0), i.e. when a negative transition of the condition occurs. The Negative Transition Filter selects which negative transitions are reported in the Event Register.

The negative and positive transition filter registers can be programmed and read with the NTRansition and PTRansition commands and queries of the OPERation and QUEStionable status model. Reading a filter register does not affect its contents.

Event Register

Setting a particular bit in both the Negative and Positive Transition Filter, causes any change (event) of the corresponding condition to be reported in the Event Register.

The contents of the Operation and Questionable Event registers can be read with the STATus:OPERation:EVENt? and STATus:QUESTionable:EVENt? queries respectively. Reading an event register will automatically clear it. Event registers cannot be programmed; i.e. there are no commands to set their contents.

Event Enable Register

Valid events can be enabled to be summarized and reported by means of the Event Enable register.

The bits in the Event Enable register define which corresponding events, recorded in the Event register, must be summarized in the Status Byte register

Operation Condition events are reported in bit 7 (OPER) of the Status Byte register.

Questionable Condition Events are reported in bit 3 (QUES) of the Status Byte register.

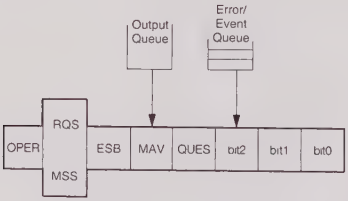
Chapter 6
Error and Event Reporting

Structure 6-2
Errors 6-4
- Command Errors 6-4
- Execution Errors 6-9
- Device dependent Errors 6-12
- Query Errors 6-13

Structure

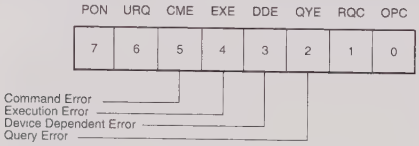
The analyzer will place detected errors and/or events in its Error/Event Queue. This queue is a FIFO (First-In-First-Out) buffer. When the queue is read, the error that occurred first, will come out first. The error that occurred last, will come out last.

When an error occurs, the corresponding error message is placed in the Error/Event queue and bit 2 of the Status Byte register is set true. Also the corresponding error bit in the Standard Event Status register is set true. If this bit is enabled (*SRE command), the GPIB controller program will be interrupted on occurrence of this error (Service Request on GPIB).



Status Byte Register

An instrument reports the occurrence of an error in the Standard Event Status Register. Each group of errors is summarized in one bit.



Standard Event Status Register

An error or event is read with the :STATus:QUEue[:NEXT]? query or its alias :SYSTem:ERRor?

Example: → :SYSTem:ERRor?
← -102, "Syntax error"

When an error is queried, the error message is transferred from the Error/Event Queue to the Output Queue. This will cause bit 4 (MAV) of the Status Byte register to be set true. The query returns the error number followed by the error description. The Error/Event and the Output Queue are FIFO buffers. First the oldest error will be read. The latest error will be read last.

If more than one error occurred, the query will return the error that was first placed in the Error/Event Queue. The error is removed from the Error/Event Queue and the next error may be read by repeating the query. When all errors are read, the queue is empty and the error message **0, "No error"** will be returned.

When errors occur while the user does not read these errors, the Error/Event Queue may overflow. Then the last error will be overwritten with the error message **-350, "Queue overflow"**. Subsequent errors will be discarded until there is space again in the Error/Event Queue.

After *RST, only errors will be reported in the Error/Event Queue. The events will only be reported, if they are enabled using the STATus:QUEue:ENABle command. A *RST command does not affect the contents of the Error/Event Queue.

The error/event queue is cleared in any of the following cases:

- Upon power up.
- Upon receipt of a *CLS command.
- Upon reading the last item from the queue.

Errors

There are four groups of standardized errors that are reported in the Standard Event Status and in the Error/Event Queue, namely:

ERROR CLASS EVENT	RANGE OF ERROR NUMBERS	STANDARD REGISTER
Command Error	-100 .. -199	bit 5 - CME
Execution Error	-200 .. -299	bit 4 - EXE
Device-Specific Error	-300 .. -399	bit 3 - DDE
Query Error	-400 .. -499	bit2 - QYE

Note: Negative Error numbers are defined by the SCPI standard. Error numbers in the ranges -300 to -399 or 1 to 32767 are defined as device-specific errors. An error value of zero indicates that no error (or event) has occurred.

Command Errors

Command Errors are errors in the range -100 to -199. They indicate that the parser has detected an error according to the syntax described in the IEEE-488.2 standard. The occurence of any command error will cause the command error bit (bit 5) to be set in the Standard Event Status register.

See also: IEEE-488.2 Section 11.5.1.

-100 Command error

This is the generic syntax error. It is given when the parser cannot detect a more specific syntax error. This code indicates only that a Command Error as defined by the IEEE standard has occurred.

See also: IEEE-488.2 Section 11.5.1.1.4.

-101 Invalid character

A syntactic element contains a character which is invalid for that element. For example, a header contains an ampersand: SYSTEM&:KEY.

This error might be given in place of errors -114, -121, -141 and some others.

-102 Syntax error

An unrecognized command or data type was encountered..

-103 Invalid separator

The parser was expecting a separator and encountered an illegal character.

-104 Data type error

The parser recognized a data element which differs from the one expected. For example, numeric data was expected while string data was received.

-105 GET not allowed

A Group Execute Trigger was received within a program message.

See also: IEEE-488.2 Section 7.7.

-108 Parameter not allowed

More parameters were received than expected. For example, the *ESE common command accepts only one parameter. So *ESE 0,1 is not allowed.

-109 Missing parameter

Fewer parameters were received than required. For example, the *ESE common command requires one parameter. So receiving *ESE is not allowed.

-110 Command header error

This error, as well as errors -111 through -114, is generated when parsing a command header. Error -110 is given when the parser cannot detect the more specific command header error.

-111 Header separator error

A character which is not a legal header separator was encountered while parsing the header. For example, no white space followed the header.

-112 Program mnemonic too long

The header contains more than 12 characters.

See also: IEEE-488.2 Section 7.6.1.4.1.

-113 Undefined header

The header is syntactically correct, but is not supported by the analyzer.

-114 Header suffix out of range

The value of a numeric suffix, attached to a program mnemonic, makes the header invalid.

See also: SCPI 1992 Volume 1 Section 6.2.5.2.

-120 Numeric data error

This error, as well as errors -121 through -129, is generated when parsing a numeric data element, including the non-decimal numeric types. This error is given if the parser cannot detect a more specific numeric data error.

-121 Invalid character in number

An invalid character was encountered in the data type being parsed.

For example, a '9' in octal data.

-123 Exponent too large

The magnitude of the exponent is larger than 32000.

See also: IEEE-488.2 Section 7.7.2.4.1.

-124 Too many digits

The mantissa of a decimal numeric data element consists of more than 255 digits excluding leading zeros.

See also: IEEE-488.2 Section 7.7.2.4.1.

-128 Numeric data not allowed

A legal numeric data element is received, but the parser does not expect this type according to the command syntax.

-130 Suffix error

This error, as well as errors -131 through -139, is generated when parsing a suffix. Error code -130 is given if the parser cannot detect a more specific suffix error.

-131 Invalid suffix

The suffix is not according to the syntax described in the IEEE standard, or the suffix is inappropriate for the analyzer.

See also: IEEE-488.2 Section 7.7.3.2.

-134 Suffix too long

The suffix consists of more than 12 characters.

See also: IEEE-488.2 Section 7.7.3.4.

-138 Suffix not allowed

A suffix was encountered after a numeric element which does not allow suffixes.

-140 Character data error

This error, as well as errors -141 through -149, is generated when a character data element is being parsed. Error -140 is given if the parser cannot detect a more specific character data error.

-141 Invalid character data

Either the character data element contains an invalid character or the particular element received is not a valid one for the command being parsed.

-144 Character data too long

The character data element consists of more than 12 characters.

See also: IEEE-488.2 Section 7.7.1.4.

-148 Character data not allowed

A legal character data element was encountered but not expected by the parser.

-150 String data error

This error, as well as errors -151 through -159, is generated when parsing a string data element. Error -150 is given when the parser cannot detect a more specific string data error.

-151 Invalid string data

A string data element was expected by the parser, but was invalid for some reason. For example an END message was received before the closing quote character.

See also: IEEE-488.2 Section 7.7.5.2.

-158 String data not allowed

A string data element was encountered but was not expected by the parser according to the syntax.

-160 Block data error

This error, as well as errors -161 through -169, is generated when parsing a block data element. Error -160 is given when the parser cannot detect a more specific block data error.

-161 Invalid block data

A block data element was encountered, but was invalid for some reason. For example, an END message was detected before the number of bytes defined by the block length, were received.

See also: IEEE-488.2 Section 7.7.6.2.

-168 Block data not allowed

A legal block data element was encountered but was not expected by the parser at this point according to the command syntax.

-170 Expression error

This error, as well as errors -171 and -178, is generated when parsing an expression data element. Error -170 is given if the parser cannot detect a more specific expression error.

-171 Invalid expression

The expression data element was invalid. For example, unmatched parenthesis.

See also: IEEE-488.2 Section 7.7.7.2.

-178 Expression data not allowed

A legal expression data element was encountered but not expected by the parser at this point according to the command syntax.

Execution Errors

Execution Errors are errors in the range -200 to -299. They indicate that an error has been detected in the analyzer's execution control block. The occurrence of any execution error will cause the execution error bit (bit 4) to be set in the Standard Event Status register.

See also: IEEE-488.2 Section 11.5.1.

-200 Execution error

This is the generic execution error. It is given when the analyzer cannot detect a more specific execution failure. This code indicates only that an Execution Error as defined by the IEEE standard has occurred.

See also: IEEE-488.2 Section 11.5.1.1.5.

-213 Init ignored

A request for initiating a measurement was ignored because another measurement was still in progress.

-220 Parameter error

This error, as well as errors -221 through -229, is generated when a program data related error occurred. Error -220 is given when the analyzer cannot detect a more specific parameter error.

-221 Settings conflict

A legal program data element was parsed but could not be executed due to the current analyzer state.

See also: IEEE-488.2 Sections 6.4.5.3. and 11.5.1.1.5.

-222 Data out of range

A legal program data element was parsed but could not be executed because the value was outside the legal range defined for the command.

See also: IEEE-488.2 Section 11.5.1.1.5.

-223 Too much data

A legal program data element of block, expression or string type was received that contained more data than the analyzer could handle due to memory or related device-specific requirements.

-224 Illegal parameter value

An exact value, from a list of possible values,was expected but not received.

-230 Data corrupt or stale

The referenced data is invalid or not present.

-232 Wrong format

The referenced data has a wrong format.

-233 Wrong version

The referenced data has the correct format but the version (number) is wrong or not supported.

-240 Hardware error

This error, as well as errors -241 through -249, is given when a legal program command or query cannot be executed because of a hardware problem in the analyzer. Error -240 is given if the analyzer cannot detect a more specific hardware error.

-241 Hardware missing

A legal program command or query cannot be executed because the required hardware is missing.

-250 Mass storage error

This error, as well as errors -251 through -259, is generated when a mass storage (diskette) error has occurred. Error -250 is given when the analyzer cannot detect a more specific diskette error.

-252 Missing media

A legal program command or query could not be executed because no diskette was installed.

-253 Corrupt media

A legal program command or query could not be executed because the diskette is corrupt or has a wrong format.

-253 Corrupt media; read error

A legal program command or query could not be executed due to a read error on the diskette.

-253 Corrupt media; write error

A legal program command or query could not be executed due to a write error on the diskette.

-254 Media full

A legal program command or query could not be executed because there was not enough room on the diskette.

-255 Directory full

A legal program command or query could not be executed because there was no free entry in the diskette directory.

-256 File name not found

A legal program command or query could not be executed because the given file name could not be found on the diskette. For example,a command was given to copy a non-existent file.

-257 File name error

A legal program command or query could not be executed because the specified file name was in error.

-258 Media protected

A legal program command or query could not be executed because the diskette was write protected.

Device Specific Errors

Device specific errors are in the range -300 to -399 or 1 to 32767. They indicate that the instrument has detected an error which is not a command error, query error or an execution error. The occurrence of any device specific error will cause the device-specific error bit (bit 3) to be set in the Standard Event Status register.

See also: IEEE-488.2 Section 11.5.1.

-300 Device-specific error

This is the generic device dependent error. It is given when the analyzer cannot detect a more specific error. This code indicates only that a Device-Dependent Error as defined by the IEEE standard has occurred.

See also: IEEE-488.2 Section 11.5.1.1.6.

-350 Queue overflow

There was no room in the queue to report an error. This code is entered into the queue in lieu of the code that caused the error.

Query Errors

Query Errors are errors in the range -400 to -499. They indicate that the output queue control of the instrument has detected a problem with the message exchange protocol described in IEEE-488.2 Chapter 6. The occurence of any query error will cause the query error bit (bit 2) to be set in the Standard Event Status register.

See also: IEEE-488.2 Section 11.5.1.

-400 Query error

This is the generic query error. It is given when the instrument cannot detect a more specific query error. This code indicates only that a Query Error as defined by the IEEE standard has occurred.

See also: IEEE-488.2 Sections 11.5.1.1.7 and 6.3.

-410 Query INTERRUPTED

A condition that caused an INTERRUPTED query has occurred. For example, a query followed by a DAB (data byte) before the response was completely sent.

See also: IEEE-488.2 Section 6.3.2.3.

-420 Query UNTERMINATED

A condition that caused an UNTERMINATED query has occurred. For example, the instrument was addressed to talk and an incomplete program message was received.

See also: IEEE-488.2 Section 6.3.2.2.

-430 Query DEADLOCKED

A condition has occurred that caused a DEADLOCKED query error. For example, both the input and the output buffer are full and the instrument cannot continue.

See also: IEEE-488.2 Section 6.3.1.7.

-440 Query UNTERMINATED after undefinite response

A query was received in the same program message after a query that requested an indefinite response.

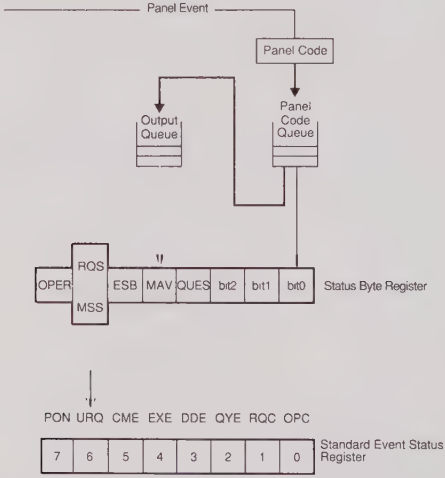
See also: IEEE-488.2 Section 6.5.7.5.

Chapter 7
Panel Event Handling

Panel Code Queue 7-2
Panel Codes 7-3
    Button Codes 7-4
    Dialknob Codes 7-5
Panel Simulation 7-6

Panel Code Queue

The following diagram indicates how front panel events are reported to the controller.



When a button is pressed or the dialknob is turned at the analyzer's front panel the following is performed:

- the corresponding keycode is placed in the Panel Code Queue.
- bit 0 of the Status Byte Register is set true. In fact bit 0 reflects the empty/non-empty state of the Panel Code Queue.
- bit 6 (URQ) of the Standard Event Status (ESR) register is set.
- if bit 6 of the standard Event Status Enable (ESE) register is set (URQ is enabled), a service request on the GPIB is generated.

The Panel Code Queue is a FIFO (First-In-First-Out) buffer which can hold a maximum of 32 key codes. An entry can be read from the queue with the :SYSTem:KEY? query. When a code is queried, the oldest code is transferred from the Panel Code Queue to the Ouput Queue. This will cause bit 4 (MAV) of the Status Byte register be set.

The :SYSTem:KEY? query will return -1 if the Panel Code Queue is empty.

The Panel Code Queue may overflow. In this case first the oldest code is removed from the queue before the new one is written in it.

The *CLS command does not affect the contents of the queue. A *RST command empties the queue.

Panel Codes

A panel code consists of 8 bits with the following layout and meaning:

bit 7 = 0: 8 bits code for panel button

bit 6 : 0 = No <shift> simultaneously pressed.

1 = <shift> simultaneously pressed.

bit 0 - 5: Button code. See table below.

bit 7 = 1: 8 bits code for dialknob

bit 6 : 0 = No <shift> simultaneously pressed.

1 = <shift> simultaneously pressed.

bit 0 - 5: Specifies the number of clicks. See table below.

Button Codes

DEC	HEX	INSCRIPTION	DEC	HEX	INSCRIPTION
0	00	FORMAT	32	20	A
1	01	I/O	33	21	E
2	02	PRINT	34	22	I
3	03	STOP	35	23	M
4	04	CONFIG	36	24	Q
5	05	TRACE	37	25	U
6	06	DISPLAY	38	26	Y
7	07	RUN	39	27	SHIFT
8	08	D	40	28	9
9	09	H	41	29	6
10	0A	L	42	2A	3
11	0B	P	43	2B	+/-
12	0C	T	44	2C	HOME
13	0D	X	45	2D	Not used
14	0E	/	46	2E	> (cursor right)
15	0F	<- (backspace)	47	2F	Not used
16	10	C	48	30	8
17	11	G	49	31	5
18	12	K	50	32	2
19	13	O	51	33	.
20	14	S	52	34	DELETE
21	15	W	53	35	^ (cursor up)
22	16	(underscore)	54	36	SELECT
23	17	INS	55	37	v (cursor down)
24	18	B	56	38	7
25	19	F	57	39	4
26	1A	J	58	3A	1
27	1B	N	59	3B	0
28	1C	R	60	3C	INSERT
29	1D	V	61	3D	Not used
30	1E	Z	62	3E	< (cursor left)
31	1F	SPACE	63	3F	Not used

Dialknob Codes

DEC	HEX		DEC	HEX	
128	80	dial -32	160	A0	dial 0
129	81	dial -31	161	A1	dial +1
130	82	dial -30	162	A2	dial +2
131	83	dial -29	163	A3	dial +3
132	84	dial -28	164	A4	dial +4
133	85	dial -27	165	A5	dial +5
134	86	dial -26	166	A6	dial +6
135	87	dial -25	167	A7	dial +7
136	88	dial -24	168	A8	dial +8
137	89	dial -23	169	A9	dial +9
138	8A	dial -22	170	AA	dial +10
139	8B	dial -21	171	AB	dial +11
140	8C	dial -20	172	AC	dial +12
141	8D	dial -19	173	AD	dial +13
142	8E	dial -18	174	AE	dial +14
143	8F	dial -17	175	AF	dial +15
144	90	dial -16	176	B0	dial +16
145	91	dial -15	177	B1	dial +17
146	92	dial -14	178	B2	dial +18
147	93	dial -13	179	B3	dial +19
148	94	dial -12	180	B4	dial +20
149	95	dial -11	181	B5	dial +21
150	96	dial -10	182	B6	dial +22
151	97	dial -9	183	B7	dial +23
152	98	dial -8	184	B8	dial +24
153	99	dial -7	185	B9	dial +25
154	9A	dial -6	186	BA	dial +26
155	9B	dial -5	187	BB	dial +27
156	9C	dial -4	188	BC	dial +28
157	9D	dial -3	189	BD	dial +29
158	9E	dial -2	190	BE	dial +30
159	9F	dial -1	191	BF	dial +31

Panel Simulation

Pressing buttons and turning the dialknob at the analyzer's front panel can be simulated by the controller with the SYSTem:KEY command. This command accepts as parameter one or more button and/or dial codes as described in the tables above. For example, the command SYSTem:KEY 5, will cause the TRACE menu to be displayed on the analyzer screen.

Note that:

- codes sent with the SYSTem:KEY command, are not placed in the Panel Code queue.
- the state of the local keyboard, locked or unlocked, has no effect on the execution of a SYSTem:KEY command; i.e. panel codes are accepted and executed even when the keyboard has been locked (see SYSTem:KLOCk command).

Chapter 8
Command Synchronization

Wait to Complete 8-3
Operation Complete 8-4

Commands and Queries are normally executed in the order of receipt. This means that Execution Control passes the message to the Device Functions block in this order (refer to Chapter 2: Introduction to IEEE-488.2, Message Exchange Control Protocol). This does not necessarily mean that the analyzer will have completed each action started by a message before the Execution Control block initiates a next action.

The advantage of parallel processing is that multiple instrument tasks can be executed partly simultaneously, to increase the speed performance of the instrument. Sometimes parallel processing is not desired. In this case sequential execution has to be forced.

With respect to command execution, SCPI distinguishes two types of commands:

Sequential commands: A sequential command is a command that always finishes before the next command is executed. A command following a sequential command will not be executed, until the analyzer has completed the action started by the sequential command. So, when only sequential commands are used, the device will never perform more than one action at a time. All commands for the analyzer are sequential with the exception of the :INITialize command.

Overlapped commands: An overlapped command is a command that allows execution of subsequent commands, while the device operations initiated by that overlapped command are still in progress. A command following an overlapped command can be executed while the action started by the overlapped command is still in progress.

The :INITialize command is the only overlapped command supported by the analyzer.

Overlapped commands typically take a relatively 'long time' to complete or their completion may be unpredictable. A 'long time' is with respect to the sequential commands. Therefore it may be important to know when the overlapped command has finished. This is to enable synchronization with other device operations and with the controller. With sequential commands in the right order, synchronization problems are not expected at instrument level. This is because the instrument will always perform the related actions one after the other.

There are three basic techniques for synchronizing between the GPIB controller and the instruments:

1. Force sequential execution using the common *WAI command.
2. Wait for the response to the *OPC? query. The response is held off until an action is completed.
3. Wait for a Service Request.

Wait to Complete

The *WAI command is used after an overlapped command to force sequential execution. The *WAI command will prevent the instrument from executing any further commands or queries until it has finished the action started by the overlapped command.

Example : The Program Message INITiate; *WAI; TIME? <pmt> will show the following time line:

The Program Message Unit "TIME?" will be executed after the :INITiate operation is finished.

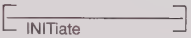
Operation Complete

Operation complete can be monitored by using the *OPC command or the *OPC? query. It should be noted that a proper use is only guaranteed, if these messages are the last units in a terminated program message.

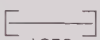
The *OPC command will set the OPC bit (bit 0) in the Standard Event Status register, when all pending operations have been completed.

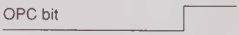
Example:

Start acquisition:



Send *OPC command:





There are two methods of synchronization between the controller and the instrument.

1. Use the OPC event bit to generate a Service Request. During the instrument operation the controller can perform other tasks if the Service Request is used as an Interrupt.
2. Read the OPC bit (bit 0) of the Standard Event Status Register using the common query *ESR?. Wait until the OPC-bit = 1.

This *OPC? query is a simple method for synchronizing the instrument operations with the controller. After the instrument responds (always with "1"), the application program can proceed knowing that the previous operation has finished.

All commands to an instrument will enter its input buffer. While these commands are fetched from the input buffer and executed by the instrument, the GPIB controller may send commands to another instrument. In this way, commands to different instruments may be executed in parallel. However, if the commands to the first instrument need to be completed before the next instrument is programmed, you may use the *OPC? to force sequential execution on the GPIB.

Example:
- Send to instrument nr 1 → :INITiate
- Send to instrument nr 1 → *OPC?
- Read from instrument nr 1 ← 1

  This response is held off until the INITiate for instrument 1 has completed. The GPIB handshake is blocked during this time.

- Send to instrument nr 2 → :SYSTem:DATE?

  This query is executed when the INITiate for instrument 1 has completed.

Notice that it is also possible to generate a Service Request when the INITIATE action is completed. For that purpose the MAV bit of the Status Byte must be enabled to generate a Service Request.

Chapter 9
Acquisition Control

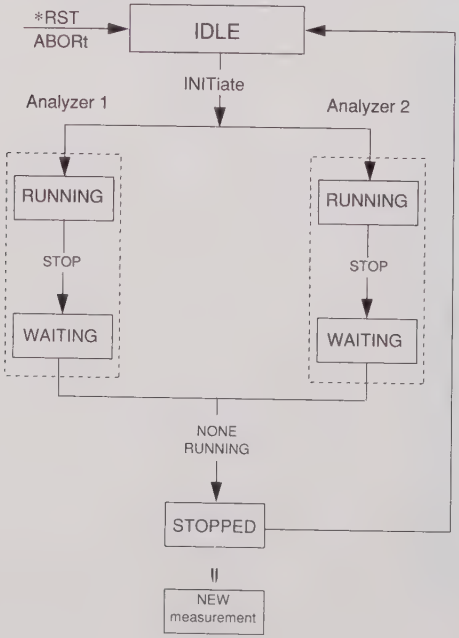
Acquisition State 9-2
Operation Condition 9-4
Synchronization 9-5

This chapter describes the facilities which are available for the remote user to control the acquisition.

The analyzer provides three commands (INITIATE, STOP and ABORT) which enable the remote controller to start and stop an acquisition. In addition, the current state (MEASURING, NOT MEASURING) is recorded in the Operation Condition register.

Acquisition State

In order to control correct execution of these facilities, the analyzer maintains internally, per logical instrument, an acquisition state (IDLE, RUNNING, WAITING and STOPPED). The following diagram shows these internal states and the transitions between them:



IDLE: The IDLE state is the only state in which a new acquisition may be initiated. The IDLE state is the initial state of the analyzer and will normally be automatically entered after an acquisition has completed. The analyzer can be forced into the IDLE state with a *RST or :ABORt command.

The :INITiate command starts the acquisition for each logical instrument which is present in the analyzer's configuration (1 for the PM3580 and 2 for the PM3585 models) and which is set "active".The logical instruments for which a measurement is initiated will enter the RUNNING state.

RUNNING: In the RUNNING state a logical instrument is acquiring data according to its current acquisition mode (STATE and/or TIMING) and its sequencer setup.

As soon as an acquisition stop occurs, either automatically according to the setup of the sequencer or as a result of a STOP command, a logical instrument enters the WAITING state.

*RST or :ABORt command aborts a running acquisition and returns the analyzer to the IDLE state.

WAITING: A logical instrument remains in the WAITING state until no (other) logical instrument is in the RUNNING state. In that case the STOPPED state will be entered.

Because there is only one logical instrument in the PM3580. a change from the RUNNING into the WAITING state is immediately followed by a transition to the STOPPED state. For the PM3585, this transition is made if the other logical instrument is in the IDLE state (no acquisition started), in the WAITING state (acquisition already stopped) or as soon as it enters the WAITING state.

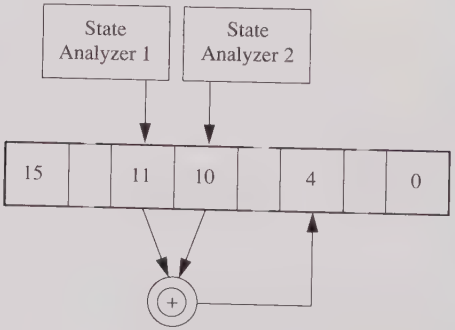
A *RST or :ABORt command returns the instrument to the IDLE state.

STOPPED: The STOPPED state is entered as soon as the acquisition for each logical instrument, for which a measurement has been started, is completed. The acquired data is copied from the acquisition memory to the NEW measurement storage and the analyzer returns to the IDLE state.

The STOPPED state is insensitive to a *RST or :ABORt command.

Operation Condition

The following diagram shows the three bits in the Operation Condition register which are used to inform the controller about the current acquisition state:



Bit 4: Measuring Summary

The value of this bit is the logical OR of bit 10 and bit 11; i.e. this bit is only 1 if at least one logical instrument is measuring.

Bit 10: Measuring Analyzer 1

This bit is 1 if the acquisition state of logical instrument 1 is RUNNING. For all other states (IDLE, WAITING and STOPPED) this bit is 0.

Bit 11: Measuring Analyzer 2

This bit reflects the acquisition state of logical instrument 2 in the same way as described for analyzer 1. For the PM3580 models this bit is always 0.

Synchronization

There are several ways for the controller to synchronize on completion of an acquisition:

1. Wait for Command Completion

A measurement is started by an INITiate command. This command is implemented as an overlapped command which completes as soon as the initiated measurement has completed; i.e. the analyzer has entered the IDLE state.

Chapter 7: Command Synchronization describes several techniques to synchronize on the completion of an outstanding (overlapped) command.

2. Read the Operation Condition Status

The current acquisition state (measuring or not) is recorded in the Operation Condition register. Because the INITiate command is implemented as an overlapped command, the controller can simultaneously read this register and wait until the measuring bit (4) becomes 0.

3. Generate Service Request

By programming the Transition Filters and Event Enable registers the controller can request the analyzer to generate a Service Request when the acquisition has completed (See also chapter 4: Status Reporting System). This is done as follows:

- Set bit 4 of the Positive Operation Transition filter to 0. This prevents an acquisition start (bit 4 of the Operation Condition register changes from 0 to 1) from being reported in the Operation Event register (bit 4).

  :STATus:OPERation:PTRansition #B.....0....

- Set bit 4 of the Negative Operation Transition filter to 1. This causes an acquisition stop (bit 4 of the Operation Condition register changes from 1 to 0) to be reported in the Operation Event register; i.e. bit 4 of this register is set to 1 when an acquisition completes.

  :STATus:OPERation:NTRansition #B.....1....

- Set bit 4 of the Operation Event Enable register to 1. This causes the Operation Summary bit (bit 7 in the Status Byte register) to be set when bit 4 of the Operation Event register changes from 0 to 1.

  :STATus:OPERation:ENABle #B.....1....

- Set bit 7 of the Service Request Enable register to 1. This causes the generation of a Service Request for the controller when the Operation Summary bit (bit 7 in the Status Byte register) changes from 0 to 1.

  *SRE #B1.....

Chapter 10
Acquisition Data Access

Measurement Structures 10-2
Memory Tables 10-5
    Table Selection 10-6
    Table Points 10-7
    Table Position 10-7
    Table Entry 10-10

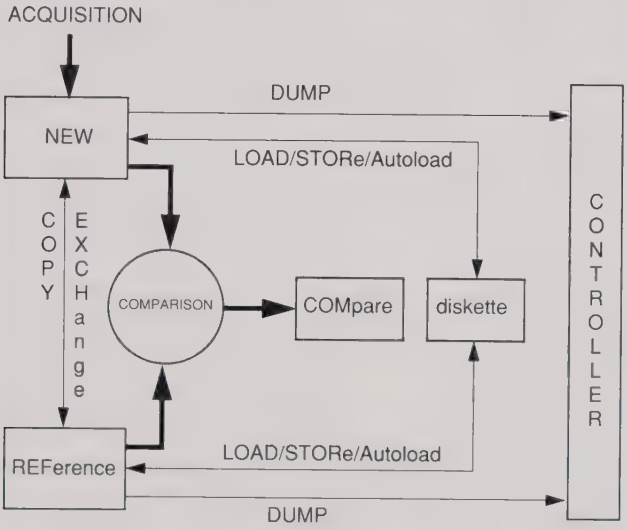
This chapter describes the facilities provided by the analyzer to enable a controller to access the acquisition data. Access is provided at two different levels:

1. Measurement access. Commands and queries which operate on a whole measurement.
2. Memory table access. Commands and queries which operate on timing, state or compare data for a particular logical instrument (Analyzer 1 or 2).

Measurement Structures

The analyzer maintains two data structures in memory, NEW and REFerence, each of which can hold the result of a single acquisition. In addition a third data structure, COMpare, contains the result of a comparison between the contents of the NEW and REFerence data structures.

The following diagram shows their relation and the flow of data between these three structures:



Flow of Measurement Data

NEW:

- At an acquisition stop (started with an INITiate command), newly acquired data is transformed and copied from specific acquisition memory to the NEW data structure.
- The command :MEMory:COPY:MEASurement can be used to copy the contents of NEW from/to the REFerence structure.
- The :MEMory:EXCHange:MEASurement command can be used to exchange the contents of the NEW and REFerence structures.

- The :MEMory:DUMP:MEASurement? query transfers the contents of NEW via the remote interface to the controller. The data is transferred in the PM 3580/ PM 3585 measurement file format; i.e. the same format as used on diskette.

- The :MMEMory:LOAD:MEASurement and the MMEMory:STORe:MEASurement commands respectively load and store the contents of NEW from/to a diskette.
  At system (re)boot an autoload file is loaded, if present on the diskette on which the SYSTEM loadfile is stored. If the autoload file contains NEW data, this data is loaded into the NEW measurement,

REFerence:
- The contents of the REF data structure is not affected by an acquisition.

- The same commands and queries described above for the NEW measurement, can be used to access the REFerence data.

COMpare:
- At any given moment, the COMpare data structure contains the result of a comparison between the contents of the NEW and REFerence data; i.e. if for some reason the contents of NEW and REFerence changes, COMpare is automatically updated.

- Commands and queries, similar to those described for the NEW and REFerence structures, to access the COMpare data are not provided. This data can only be accessed via the MEMory:TABLe commands and queries.

The contents of NEW and/or REFerence is cleared in each of the following cases:

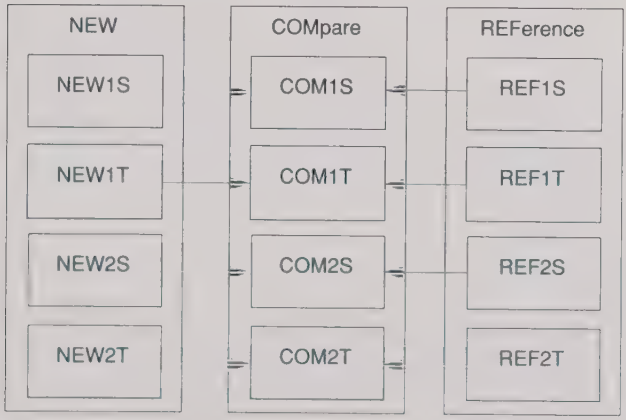
- At system (re)boot.
- When the system is reset.
- As the result of a MEMory:CLEar:MEASurement command.

Memory Tables

Each measurement structure described above, NEW, REFerence and COMpare, consists of four memory tables. A memory table is a named entity. It contains either State or Timing information for either logical instrument 1 or 2.

The NEW and REFerence tables contain the sampled channel values, status information and timestamps obtained by an acquisition. A COMpare table contains the result of a comparison between the contents of the corresponding tables in NEW and REFerence.

The following diagram shows the relation between the measurement structures and their corresponding tables:



Measurement Structures and Memory Tables

The following table shows the relation between a table name and the type of data stored in the corresponding memory table:

Name	Measurement	Analyzer	Data-Type
NEW1S	NEW	1	STATE
NEW1T	NEW	1	TIMING
NEW2S	NEW	2	STATE
NEW2T	NEW	2	TIMING
REF1S	REFerence	1	STATE
REF1T	REFerence	1	TIMING
REF2S	REFerence	2	STATE
REF2T	REFerence	2	TIMING
COM1S	COMpare	1	STATE
COM1T	COMpare	1	TIMING
COM2S	COMpare	2	STATE
COM2T	COMpare	2	TIMING

Memory Table Names

Table Selection

The analyzer allows the controller to access only one memory table at a time. In order to access a particular table, it must first be selected. This is done with the :MEMory:TABLe:SELect command. The query form of this command returns the currently selected table.

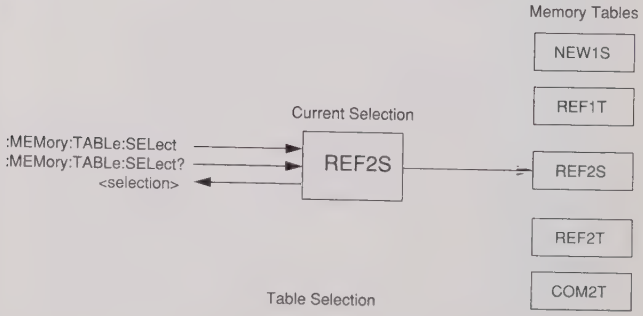


Table Selection

After a table has been selected, all subsequent :MEMory:TABLe commands and queries (except SELect) operate on this table until a new select command is given.

When a memory table access command is received (other than SELect) while no table has yet been selected (*RST condition), the analyzer selects a table according to the following algorithm:

1. Select the first table (in the order shown above in the Table Name list; i.e. NEW1S, NEW1T, NEW2S, etc.) for which data has been acquired.
2. If the previous step fails (no data acquired yet), select the first table in which data would be stored if an acquisition has been made.

Table Points

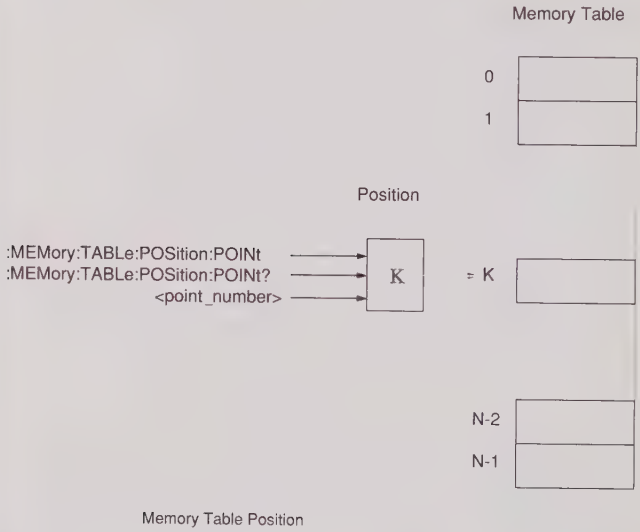
Each memory table consists of zero, one or more entries. The number of entries present in a memory table depends on the number of samples acquired (NEW and REFerence) or compared (COMpare) for that table. In general this number will differ per table.

The :MEMory:TABLe:POINts? query returns the number of entries (points) in the currently selected table. This number can not be set by the controller; i.e. there is no corresponding :MEMory:TABLe:POINts command.

Table Position

In order to provide access to each individual entry, a point (entry) number (0, 1, 2, etc.) is associated with each table entry. The analyzer maintains a variable containing the current position (point number). Requests from the controller to access the contents of a table entry, will operate on the entry that corresponds with the current point number.

The :MEMory:TABLe:POSition:POINt command allows the controller to explicitly set the position. The query form of this command returns the current point number.



Memory Table Position

As an alternative to positioning on points, the analyzer provides three other :MEMory:TABLe:POSition commands which allow the controller to specify the required position in a different way. In these cases, the specified position is transformed into a point number which is taken as the current position. The query form of these commands perform the opposite; i.e. the current point number is transformed and returned to the controller. The alternatives are:

- The :MEMory:TABLe:POSition:NUMBer command allows the specification of a sample number as required position. Sample numbers are numbers relative to the sample at which the Trigger occurred. The sample number of the Trigger sample is 0.

  For example, suppose the current POINt position is 12 and the trigger position in the currently selected memory table is at POINt number 14.

- The query :MEMory:TABLe:POSition:POINt? returns 12.
- The query :MEMory:TABLe:POSition:NUMBer? returns -2 (i.e. 12-14).
- The command :MEMory:TABLe:POSition:NUMBer 18 sets the current position to POINT number 32 (i.e. 14+18).

- The :MEMory:TABLe:POSition:STAMp command allows the specification of a time stamp as required position. Timestamps are relative to T_o. T_o has the value 0 and is the timestamp which is assigned to the oldest Trigger point or, if not present, with the oldest sample in any of the memory tables. One reference timestamp (T_o) allows all timestamped data, present in any memory table, to be correlated on basis of time.

  For example, suppose the current POINT number is 12 and the timestamps of points 12, 13 and 14 in the currently selected memory table are respectively 370, 410 and 560:

  - The query :MEMory:TABLe:POSition:STAMp? returns 370.
  - The command :MEMory:TABLe:POSition:STAMp 420 sets the current position to POINT number 13 (420 is between 410 and 560).

- The :MEMory:TABle:POSition:VALue command allows the specification of a (symbolic) value as required position. The currently selected memory table is searched for an entry with the specified sample value and, if found, the current position is set to the corresponding entry number.

All position commands and queries, with the exception of positioning on a value, accept the following names as a symbolic specification of the required position:

MINimum:	Represents the first entry in the table; i.e. point number 0.
MAXimum:	The last entry in the table.
TRIGger:	The entry that corresponds with the trigger point.

For example, the command :MEMory:TABLe:POSition:POINt TRIGger will set the current position to the point number that is associated with the trigger sample. The query form of this command returns the trigger point number.

Table Entry

Each memory table entry contains the data of one acquired or compared sample. This data consists of three elements: the time stamp, the sample value and the sequencer status.

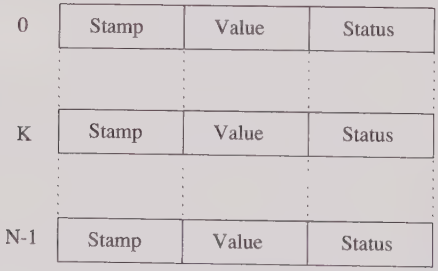


Table Entries

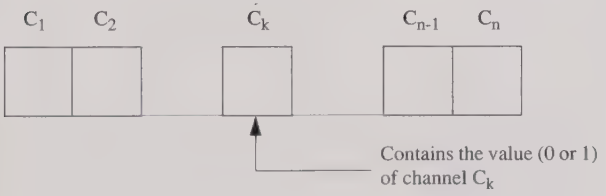
The analyzer allows the controller to retrieve the three elements of an entry together (:MEMory:TABLe:SAMPle:DATA?) or individually (:MEMory:TABLe:SAMPle:XXXX?). In the latter case, the name of the element is the leaf node of the query header; e.g. the query :MEMory:TABLe:SAMPle:STATus? is used to obtain the status.

By default, the queries that access the contents of table entries operate on one entry at the time; i.e. the current entry. The controller however may specify how many entries should be accessed via an optional query parameter. In this case, the requested data of the current and subsequent entries is returned. After query completion, the current position is incremented/decremented by the actual number of entries accessed.

The direction in which the table entries must be accessed, either forward or reversed direction, can be specified with the :MEMory:TABLe:ACCess:FORWard command.

Sample Value

All sample value fields in a given memory table are of equal size and consist of n bits; one bit for each channel that was assigned, and for which data has been captured during the previous acquisition.



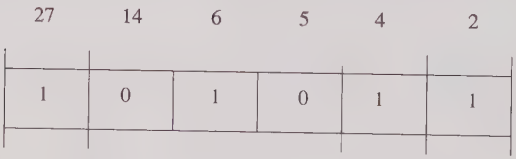
Sample Value

The query :MEMory:TABLe:CHANlist? returns a channel list expression, showing for which channels data is stored in the currently selected memory table. This channel list is taken as the default for those commands and queries that have an optional channel list parameter.

The query :MEMory:TABLe:SAMPle:VALue? returns one integer value, representing the channel values stored in the current entry of the currently selected memory table.
By default, the returned integer value is composed of all channels for which data is recorded in the table. The controller may however specify, via an optional query parameter, which channels should be taken into account. The order in which the channels are specified defines the significance of their corresponding bit values; i.e. the first channel specified is the most significant one.
A value of 0 is returned for a channel, not recorded in the memory table.

The command :FORMat:DATa can be used to select the output format; i.e. ASCii, BINary, etc.

For example, assume that in the currently selected memory table data has been stored for channels 2, 4, 5, 6, 14 and 27. Furthermore assume that the channel values in the current entry are 1, 1, 0, 1, 0 and 1 respectively. This is visualized in the following picture:



Example: Sample Value

- The query :MEMory:TABLe:CHANlist? will return the expression (@27,14,6:4,2)
- The query :MEMory:TABLe:SAMPle:VALue? will return 43, #B101011, #Q73 or #H2B when ASCii, BINary, OCTal or HEXadecimal output format respectively is selected.
- The query :MEMory:TABLe:SAMPle:VALue? (@27,5,2) will return the ASCii value 5 (BINary: #B101).
- The query :MEMory:TABLe:SAMPle:VALue? (@5,2,27) will return the ASCii value 3 (BINay: #B011).
- The query :MEMory:TABLe:SAMPle:VALue? (@27,5,2,0) will return the ASCii value 10 (BINary: #B1010).

Time Stamp

The query :MEMory:TABLe:SAMPle:STAMp? returns the timestamp of the current entry in the currently selected memory table.
The format and length of this response can be selected with the :FORMat:STAMp command.

The query :MEMory:TABLe:SAMPle:DURation? returns the duration of the current entry in the currently selected memory table. This duration is defined as the timestamp of the current sample minus the previous one.
The format and length of the response can be selected with the :FORMat:DATA command.

Both timestamps and durations are given in units of 5ns.

Sample Status

The query :MEMory:TABLe:SAMPle:STATus? returns the sequencer status of the current entry in the currently selected memory table. The response is an integer in the range 0 to 8, with the following meaning:

0 = Sample taken at sequence level 1

1 = Sample taken at sequence level 2

2 = Sample taken at sequence level 3

3 = Sample taken at sequence level 4

4 = Sample taken at sequence level 5

5 = Sample taken at sequence level 6

6 = Sample taken at sequence level 7

7 = Sample taken at sequence level 8

8 = Sample taken after trigger

Chapter 11
Remote/Local Operation

Remote Operation 11-2
Local Operation 11-3
Operator Communication 11-4

The analyzer does not support the Remote/Local requirements defined in the IEEE-488.2 standard section 5.6. All possible Remote/Local state transitions described in that document are ignored by the analyzer.

If both remote and local operation are enabled, commands and queries from the controller are executed concurrently with front panel input. In this case, special care must be taken because dual operation may lead to unwanted situations for both the remote and the local site. For example, when both parties are changing the same analyzer settings.
In order to avoid any interference, facilities are provided that allow the controller, as well as the local operator to obtain exclusive control over the instrument.

Remote Operation

If local operation has not been disabled, the remote commands and queries are executed in parallel with front panel input. In this case provisions have been taken to avoid conflicts. The execution of a command or query received over the remote interface, might be suspended under the following circumstances:

- The local operator is executing a dialogue; e.g. selecting an item from a list popup, editing a field, answering to a question (popup), etc. Whether a dialogue is busy or not, is reflected in bit 8 of the Operation Condition register. See Chapter 5: Status Reporting System.
- The diskette is in use for the execution of a local request. For example, a measurement file is being loaded.
- The acquisition is running. The acquisition state is recorded in the Operation Condition register. See Chapter 9: Acquisition State.

If the execution of a particular command can be suspended for one of the reasons mentioned above, this is explicitly described in the appropriate command description in Chapter 4.

Note that only the execution is suspended. This means that the command or query is first completely parsed and possible parse errors will be reported immediately.

In order to obtain exclusive access to the analyzer, the controller must send the command :SYSTem:KLOCk ON to disable front panel input. The only way in which local operation can be re-established again is either by rebooting the system or by sending the command :SYSTem:KLOCk OFF. In addition, the local screen can be switched off and on with the :DISPlay:ENABle command.

Local Operation

If remote operation has been installed and enabled, analyzer's front panel input will be handled concurrently with the execution of commands and queries received from the controller. In this case the local operator must be very carefull when changing the analyzer settings or initiating an acquisition. He can freely inspect the settings and analyzer the acquisition data.

In order to obtain exclusive control over the analyzer, the local operator has to switch off the remote interface in the I/O menu; i.e. Remote: Off. Refer to the PM 3580/PM 3585 Reference Guide: I/O Menu for more information. A request to switch off the remote port is rejected (Error popup is displayed) if the remote interface is currently busy; i.e. a command or query has (partly) been received and the execution has not yet completed.

Remote operation can only be restored by selecting a remote port in the I/O menu. Note that remote port selection is saved in battery back-up RAM, so after a system reboot the current selection is restored.

Operator Communication

The analyzer provides several commands and queries which allow the controller to communicate with the local operator. In this case the initiative is with the controller, who can request something from the local operator. If such a command or query is received while local operation is disabled, local operation is temporarily enabled (and if necessary the display is temporarily switched on) until the request has been honored. For a description of the commands, refer to Chapter 4: Display Subsystem.

If the local operator requires the attention of the controller, he can press any button on the analyzer's front panel or turn the dialknob. This causes the corresponding panel code to be placed in the Panel Code queue and bit 6 to be set in the standard Event Status register. Refer to Chapter 7: Panel Event Handling for a detailed description. Note that if local operation is enabled, pressing a button will also be regarded as regular front panel input.

Chapter 12
RS-232 Interface

Analyzer Setup 12-2
Interface 12-3
- Signals 12-3
- Cabling 12-4
- Data Format 12-5
- Handshaking 12-6
- Data Encoding 12-6

Limitations 12-7

This chapter only applies to product PF8653/30: Remote Control via the RS-232 interface. It describes instrument set up, cabling, transmission characteristics and some limitations for remote operation via the RS-232 interface.

Analyzer Setup

The RS-232 interface of the analyzer is factory installed. Before the analyzer can be programmed by a controller via the RS-232 interface, the following has to be performed:

- **Install RS-232 software.**

  Some additional RS-232 software needs to be installed. This software is present on the diskette supplied with this product (PF8653/30). In order to install this software, the following procedure should be followed:

  1. Remove any diskette currently inserted in the diskette drive.
  2. Switch the power of the analyzer off and on, or reboot.
  3. Insert the installation diskette in the diskette drive.
  4. Follow the instructions displayed by the RS-232 Installation Utility program.

- **Select the RS-232 interface.**

  After the RS-232 interface has been installed, the system software (PF8690/xx) has to be set up to accept commands via the RS-232 interface. This is done as follows:

  1. Switch to the I/O menu.
  2. Select the field labeled "Remote:", choose the option RS-232 and close the popup.

- **Select the RS-232 transmission settings.**

  In the I/O menu, the RS-232 settings (baud rate, parity, etc.) can be changed if the default settings are not correct.

The installed software, the selected RS-232 port and the RS-232 settings are saved in battery-backup RAM. After a system boot the system is automatically configured in accordance with the last selected settings.

Interface

The RS-232C standard describes the communication between data terminal equipment (DTE) and data communications equipment (DCE). The standard does not describe the communication between two DTE's (e.g. two computers). Additionaly, the handshaking signals defined in this standard do not provide sufficient functionality to make reliable high speed communication possible. In order to overcome these problems, the signals for the analyzer have been modified.

Signals

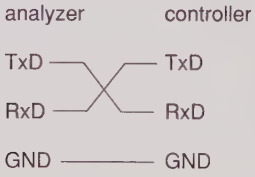
The following signals, which is a subset of the signals defined by RS-232C, are used on the RS-232 port of the analyzer:

pin nr	signal	function
1	GND	Protective ground. This pin is connected with pin 7.
2	TD	Transmitted Data. This pin transmits the data from the analyzer to the controller.
3	RD	Received Data. This pin transmits the data from the controller to the analyzer.
4	RTS	Request To Send. With this signal the analyzer states that it is ready to receive data on the RD line.
5	CTS	Clear To Send. The analyzer uses this signal to

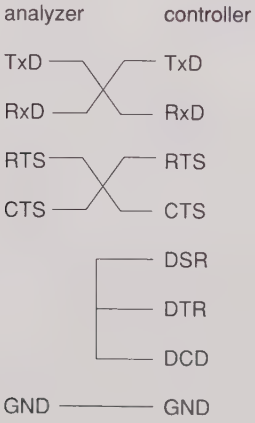
		determine if it may send data via the TD line. When hardware handshaking is disabled, the state of this line is ignored.
7	GND	Signal ground. This pin is connected with pin 1.

Cabling

When no handshaking is used, the analyzer can be connected to the controller with the following cable:



When hardware handshaking is used, the following cable must be used. Note that the signals DSR, DTR and DCD are not used on the instrument's site.



Data Format

The format of the data, transferred via the TD and RD pins, is defined by the baud rate, the parity, the number of data bits and stop bits.

These parameters can be set in the Input/Output menu. Refer to the Reference Guide.

Baud Rate

The baud rate determines the rate at which data bits are tranported.

It must be equally setup for the analyzer and the controller.

The supported baud rates are: 75, 150, 300, 1200, 2400, 4800, 9600 and 19200.

Parity

The parity (bit) is used to detect errors in order to increase the reliability of data transmission over serial communication lines. With this method, a single-bit error in a transmitted byte can be detected.

Parity can be set OFF, EVEN (meaning the total number of 1-bits in a transmitted data byte plus the parity bit is even) or ODD.

Data Bits

Data bytes are transmitted as a serie of n bits, with n being 7 or 8.

Note that some commands and queries require the transfer of 8 bit (binary) data. So if the number of data bits has been set to 7, these commands will not work properly.

In case of 7-bit incoming data, the analyzer will set the most significant bit of each received byte to zero.

Stop Bits

After the transmission of the data and parity bits, either one or two stop bits are transmitted to mark the end of the transmitted byte.

SCPI defines 1 stop bit for outgoing data and any number of stop bits for incoming data. Even if the number of stop bits has been set to 1, the analyzer will accept incoming data with 2 stop bits.

Handshaking

The analyzer can either operate without handshaking or with the RTS/CTS protocol. The latter is implemented as follows:

- Before transmitting a byte, the analyzer examines the state of the CTS line (which is connected to the RTS line of the controller). If this line is asserted, transmission of the byte is started, otherwise it is delayed.
- As long as the analyzer can receive data, it asserts the RTS line (which is connected to the CTS line of the controller). When the input buffer is nearly full the analyzer negates its RTS line as an indication for the controller that it should suspend transmission of data.

Although the analyzer allows handshaking to be switched off, the SCPI standard states that handshaking is required.

An other common implementation is Xon/Xoff software handshaking. This is not supported by the analyzer because these flow control characters cannot be distinguished when binary data is being transferred.

Data Encoding

The IEEE interface has in addition to its normal data path, some hardware extensions to allow special commands to be transmitted (even in the middle of data exchange). The RS-232 interface does not have these signals, so they have to be sent as special character sequences.

Normal Data

Special characters are transmitted as a two character sequence of which the first one is an ESC (escape) character (hexadecimal 1B). In order to distinguish a regular ESC character from the special one, it should be transmitted twice (ESC-ESC).

In order to make input and output behaviour the same, the analyzer will also transmit two ESC sequences for one regular escape.

DCAS

The Device Clear Active State should be sent as an ESC character followed by a 4 (ESC-4).

In some cases, it is possible that data input to the analyzer is not handled for a long period of time. For example, if the input buffer of the analyzer is full and the RTS line is negated. The controller should not send any more data, but if a DCAS is to be sent, the following procedure should be followed:

- Send any character. This will cause an input overrun in the analyzer.
- Wait until RTS gets asserted again.
- Send (with the normal handshaking protocol) the sequence ESC-4.

Unknown ESC Sequences

Any escape sequence not supported by the analyzer, will be discarded.

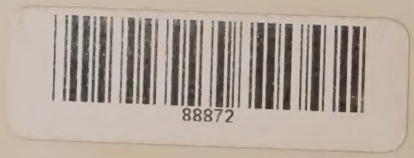
Break Character

A break character, i.e. a character with all bits equal to zero (including the stop bits, which may last several character times) is ignored by the analyzer.

Limitations

This product is based upon the SCPI and IEEE-488.2 standards. In general, the functionality, as described in the previous chapters is identical for the IEEE and the RS-232 interface. The following minor limitations and restrictions apply to remote control via RS-232:

- The program message terminator (pmt) should be a New Line (decimal 10) character.
- The response message terminator (rmt) is a New Line character.
- Service requests are not generated.
- Serial poll is not supported. As an alternative, the controller can periodically poll the Status Byte register.
- Indefinite length arbitrary blocks may not be sent to the analyzer, because block terminators cannot be distinguished from arbitrary data characters.


88872